1988

A System for the generation of tonal music based on transformations

Karl J. Myers

Rochester Institute of Technology
School of Computer Science and Technology

A System for the Generation of Tonal Music Based on Transformations

by
Karl J. Myers

A thesis, submitted to
The Faculty of the School of Computer Science and Technology,
In partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

Approved by : John A. Biles 3/2/1988
Professor John A. Biles

Approved by : Peter G. Anderson 25 Feb 88
~~Professor~~ Peter G. Anderson

Approved by : Rodger Baker
Professor Rodger Baker

January 28, 1988

I hereby grant permission to the Wallace memorial Library, of RIT, to reproduce my thesis, "A System for the Generation of Tonal Music Based on Schenkerian Transformations", in whole or in part. Any reproduction will not be for commercial use or profit.

Karl J. Myers _____ 3/21/88

ABSTRACT

Most if not all of the approaches being used in the computer study of natural languages also have been applied to the computer study of music. Music theorist Heinrich Schenker (1867 - 1935) developed a system for the study of tonal music that is compatible with the transformational grammars defined by linguist Noam Chomsky for the study of natural languages. Steven Smoliar (1980) defined transformations, based on Schenker's theory, for the analysis of musical compositions. This study provides background into the application of grammars in the generation of musical compositions and a full implementation of the transformations defined by Smoliar. The implementation is in 'C', and the transformations can be performed interactively in an interpreted mode or incorporated into any 'C' language program.

keywords: computer music, Schenker, Smoliar

TABLE OF CONTENTS

1. Introduction 1

2. Background 5

2.1 On the Nature of Tonal Music 5

2.1.1 Overview 5

2.1.2 An Overview of Schenker's Analysis 8

2.1.3 Schenker's Work as Cognitive Psychology.. 11

2.2 Previous Work 12

3. Implementation 17

3.1 Overview 17

3.2 Program Modules 17

3.3 Implementation Functionality 19

3.3.1 Data Structures 19

3.3.2 "Upper-Level" Functions 22

3.3.3 "Lower-Level" Functions 26

3.3.4 Utility Functions 35

4. Results 37

4.1 Results 37

4.2 Sample Session 37

5. Conclusions 44

6. Bibliography 49

7. Appendix 51

7.1 "C" Language Interface 51

7.2 Sample Session 57

7.3 Source Code Listing 73

1. Introduction

Heinrich Schenker (1867 - 1935), music theorist and educator, formulated a system for the study of tonal music based on the existence of deep structure within musical compositions. Schenker defined a system consisting of three levels of structure in which the surface structure for all tonal musical compositions may be reduced to a less-complex intermediate level, and this level in turn may be reduced to a less-complex proto-structure or root. There are strong parallels between this system of analysis and the transformational grammars defined by Noam Chomsky [CHOM65]. I have implemented a system for the generation of musical compositions according to the transformations used in Schenker's analysis. This system is essentially a replication of the system devised by Smoliar [SMOL80] and differs substantially from Smoliar's system only by a stricter adherence to Chomsky's definition of transformational grammars.

Transformational grammars were formulated by Noam Chomsky [CHOM65] in order to explain the structural differences between semantically similar natural language sentences. The following sentences are structurally different, but semantically equivalent:

Sylvester saw Tom.

Tom was seen by Sylvester.

Each sentence is represented by a tree-like diagram in figure 1.1.

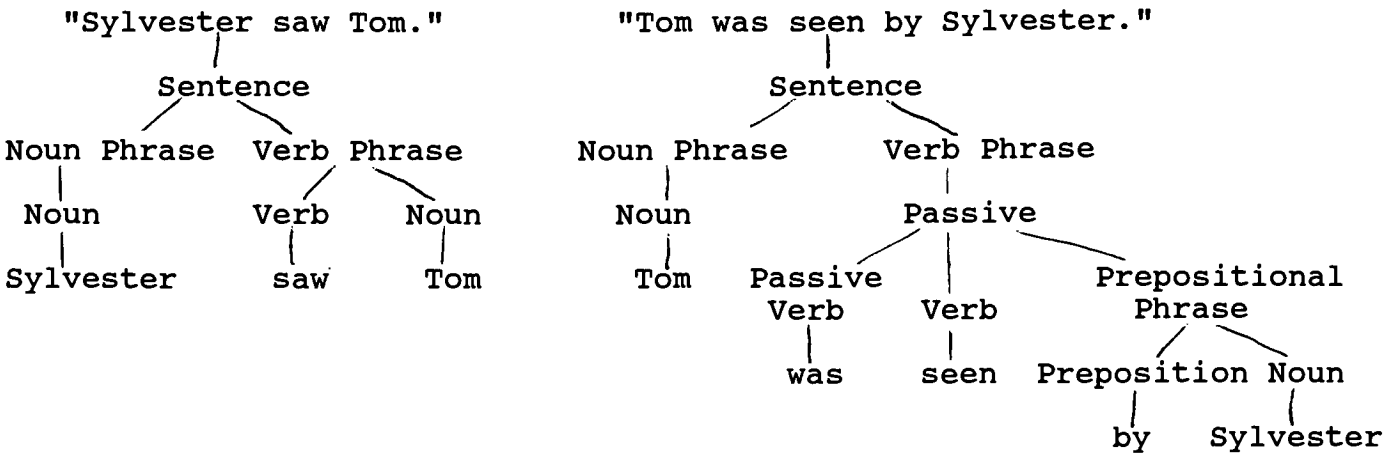


Fig. 1.1 Sentence diagrams

A transformational grammar consists of a finite number of transformations. These transformations define the changes that may be applied to entity A to produce equivalent entity B. In the above example, the sentence, "Sylvester saw Tom." is transformed into the sentence, "Tom was seen by Sylvester." by the following transformations:

1. The noun, Sylvester is transformed into the prepositional phrase "by Sylvester" with the introduction of the preposition "by". This prepositional phrase is incorporated into the new verb phrase.

2. The verb "saw" is transformed into its passive equivalent, "was seen" and remains part of the verb phrase.

3. "Tom", the noun phrase part of the verb phrase in the original sentence, is transformed into the noun phrase of the resulting sentence.

Schenkerian analysis views a composition as the surface structure of a complex system. This system consists of surface structure and deep structure. Schenker postulated the existence of musical proto-structures, which are re-written, or transformed, into surface structure. The analysis first seeks to uncover the relationships within the deep structure. When these relationships are uncovered, surface structure level notes, those that are closer to the proto-structure (are the result of fewer transformations) are recognized as more "important". These more "important" notes are stressed by the performer when playing the composition.

The generation of musical compositions is based on Schenker's work but must proceed in a manner fundamentally different from Schenker's analysis. Whereas Schenker's analysis proceeds from surface structure to deep structure, generation of a new composition will proceed from the root toward surface structure.

Chapter 2 provides the theoretical basis of the application of transformational grammars to tonal music. Heinrich Schenker's approach to the study of music and Noam Chomsky's definition of transformational grammars are discussed as is Steven Smoliar's definition of a system based on Schenker's music theory and transformational grammars.

The details of the implementation of the system for the generation of tonal compositions are discussed in Chapter 3. Chapter 4 contains examples of the use of the system implementation. Sample

sessions, including a replication of Smoliar's sample analysis, results and a discussion of those results accompany the examples. A discussion of the overall conclusions of the study can be found in Chapter 5. The appendix contains materials that users of the system will find useful. These materials include a user's guide, which describes the transformations that can be performed using the system interactively in an interpreted mode. A description of the 'C' language transformation functions that comprise the 'C' toolkit is included as is the listing of the implementation code and a sample session.

2. Background

2.1. On the Nature of Tonal Music.

2.1.1 Overview

Throughout almost the entire history of music, the tendency of compositions to give preference to one tone (called the tonic, or key note) and to relate all other tones to this primary tone has been evident. This establishment of tonality is apparent in monophonic music, which is purely melodic music lacking harmony as well as in polyphonic or fully harmonized works. Monophonic music is comprised of a single melodic line without additional parts or accompaniment. From the Greek monos meaning one and phonos meaning sound, monophonic music is probably best known in the form of Gregorian chant, the ancient liturgical music of the Roman Catholic church, characterized by unisonous melodies (in which all voices sing the same "tune") and free rhythm. Polyphonic (Greek poly = many, phonos = voices) music is written as a combination of several simultaneous voices (or parts) of a more or less pronounced individuality [APEL44]. The establishment of a tonal center is the cornerstone upon which all Western music except the "atonal" music of the twentieth century is built. Atonal music is music in which a tonal center is scrupulously avoided.

Although all music (except the just mentioned "atonal" music) is tonal, the means of achieving tonality have changed greatly over the course of music history. In monophonic music the tonality is by necessity established by the melody alone.

The stressing of certain tones or the cadence or similar events will point to a tonal center. In the field of harmonized music the situation is more complex. Melodic tonality is supported by the underlying harmony. This harmony, like melody, must be related to the tonic. Around 1700 three main chords plus one common variation, the tonic, the dominant, and the subdominant triads and the dominant seventh chord, became generally accepted as the basis for a system of tonal functions. Indicated by the numeral I in harmonic analysis, the tonic triad, or three-part chord, is built upon the key note of the piece. Its base note is the tonic note, its second member third above the tonic and its final member a third above that, or a fifth above the tonic. The dominant triad is built in the same manner based on the fifth degree of the key scale, called the dominant because of its dominating position in harmony as well as melody. The dominant seventh chord is built on the same foundation, but has an additional tone a third above the highest tone of the dominant triad, or a seventh above the fundamental. In harmonic analysis, the dominant triad is indicated by the numeral V, the dominant seventh by V7. The subdominant triad is similarly built upon the fourth degree of the key scale, deriving its name from the fact that this tone is a fifth below the tonic (as the dominant is a fifth above it.) The subdominant is indicated in harmonic analysis by the numeral IV. Common practice dictated as "usual" such events as the chord progression I-IV-V-I or the cadence V7-I but disallowed the harmonic movement I-V-IV-I.(1)

In this paper, the following identifying terminology is used:

C 0 refers to middle C, C -1 is one octave below middle C, C -2 is two octaves below, C 1 is one octave above middle C, and so on. The tones C - D - E - F - G - A - B refer to the white keys on the piano. The symbol #, or sharp, indicates the tone is to be raised 1/2 step. The symbol b, or flat, lowers the pitch 1/2 step. The piano keys from C 0 to C 1 would be represented thus:

C - C# - D - D# - E - F - F# - G - G# - A - A# - B - C.

This system of tonal functions prevailed through the eighteenth and nineteenth centuries and was extended during this time by the more frequent use of chromatic alterations involving the use of tones outside the diatonic scale (e.g. in C major C - D# - E or C - D - D# - E instead of the diatonic progression: C - D - E) and modulations into other keys by means of chords common to both the initial and new keys.

The musical theories of Heinrich Schenker are based upon this system of tonal functions. More specifically, Schenker applied his analysis to the compositions of the period from Bach to Brahms, excluding Wagner. Schenker's pupils also have applied his method to medieval music. Wagner's extremely chromatic work defies straightforward Schenkerian analysis, although it has been suggested that this analysis could be applied to late Romantic music as well as twentieth century tonal music.

---

1. The basis for disallowing the I-V-IV-I progression is the difficulty in dealing with the parallel movement of inner voices that occurs during this progression. This is often and incorrectly treated solely as a music theory issue and not a music history consideration.

2.1.2. An Overview of Schenker's Analysis.

Through stepwise reduction, Schenker's analysis leads from the surface structure of the actual composition (foreground in Schenker's terminology) through what Schenker referred to as the middleground to the background structural level. The background level is that nearest the root of the composition. Schenker devised his own notation to describe the deep structure of compositions. Examples of this notation can be found in [FRAN78]. This notation is compatible with the widely used tree diagrams. Thus, a composition can be diagramed as a tree. The leaves of the tree represent the actual composition. The root of the tree consists of a single note, the tonal center of the composition. This root is called the Ursatz in Schenker's terminology and the proto-structure in Smoliar's terminology [SMOL80]. The root is viewed as having been transformed into increasingly complex equivalent trees with the result being the actual composition.

This analysis attempts to reveal the organic structure of music by showing that every composition ultimately follows some simple structural tone pattern, which acts as its skeleton and guarantees its continuity. The analysis is concerned with the perceived deep structure of the composition. It is not the purpose of this kind of analysis to show that all the various compositions can be reduced to a few types of "fundamental structure." The analysis has to proceed from the foreground to the background, but it should be read in the opposite

direction. Schenker's analysis does not seek to prove that ultimately all compositions are more or less alike; it demonstrates how a few patterns unfold into the infinite variety of actual compositions. Its main interest does not lie in the background itself, but in the point where it shows how the background and foreground are connected. This is called the middleground and it is here that the rich complexity of the composition is clearly evident. The relation between the actual music and its Ursatz is completely subconscious to the composer and disjoint from the creative process.

Smoliar [SMO80] defined the nature, ursatz, binary and ternary transformations based upon observations of Schenker's work. These transformations shape an entire composition or subcomposition. Nature transforms a single note into a chord consisting of the first 5, 6 or 7 tones of the natural overtone series of the note. Smoliar called this resulting chord a "chord of nature." Figure 2.1 is an example of a "chord of nature."

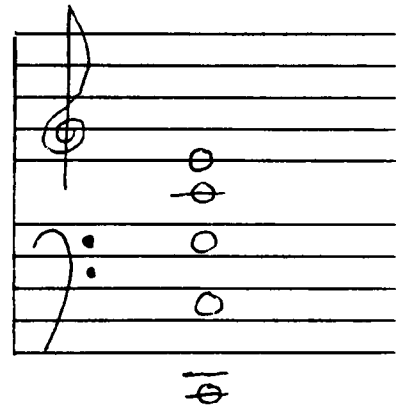


Fig. 2.1 "Chord of Nature" based on the note "C"

Ursatz transforms a "chord of nature" into its corresponding Ursatz form. This consists of a simultaneity of a descending

sequence of treble tones and tonic - dominant - tonic progression of bass tones. Figure 2.2 shows the Ursatz form corresponding to the above "chord of nature".

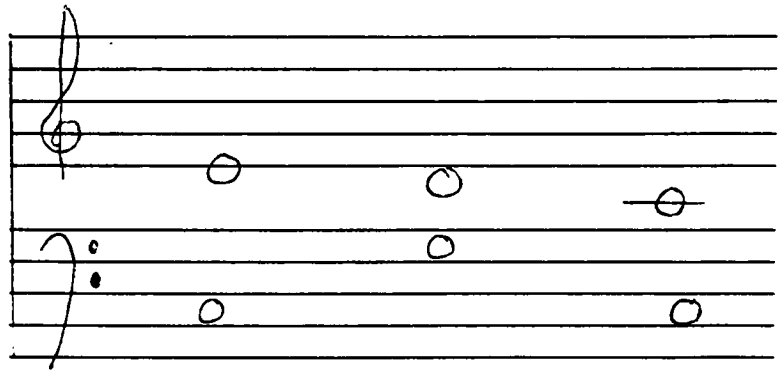


Fig. 2.2 Example of Ursatz form

Binary transforms a ursatz form by replacing the sequence of treble notes with a sequence of two sequences of these treble notes. Similarly, the sequence of bass notes is replaced by a sequence of two sequences of the bass notes. Figure 2.3 shows the Binary form corresponding to the above ursatz form.

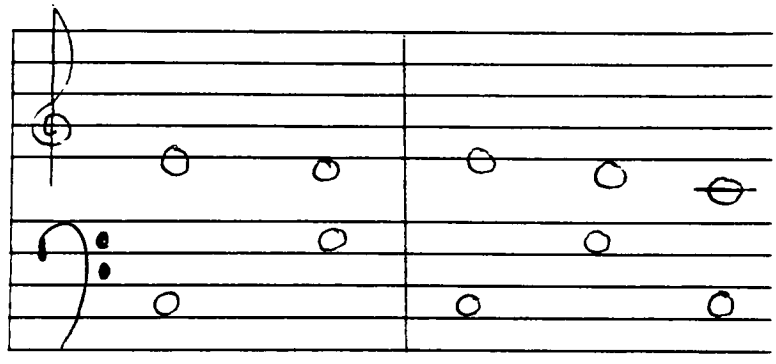


Fig. 2.3 Example of Binary form

Ternary transforms a binary form by the insertion of two notes of the dominant triad (the first and fifth of the chord) between the two sequences of treble and bass notes. Figure 2.4 shows the

Ternary form corresponding to the above Binary form.

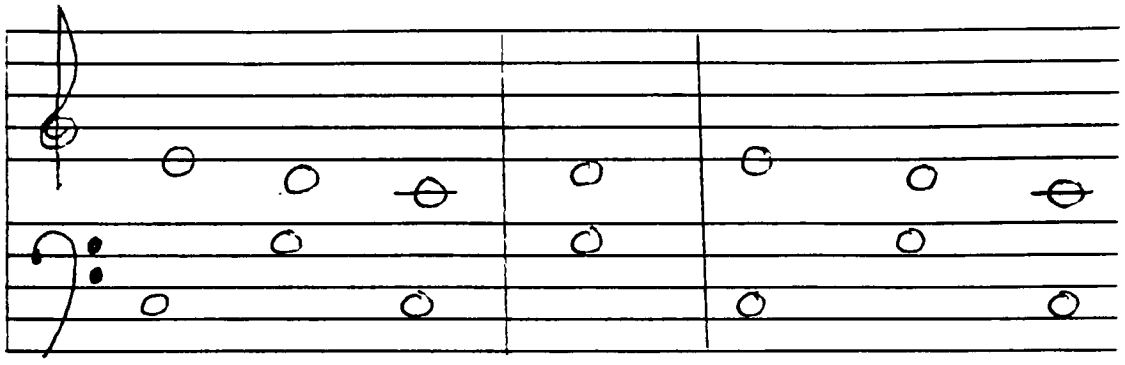


Fig. 2.4 Example of Ternary form

For an excellent thorough review of Schenker's theories, see [FORT77].

Schenker's theories are most important to computer scientists because they are compatible with tree transformations, and transformational grammars, which are well-defined. By using these transformations in a generative capacity we can create "syntactically correct" compositions.

2.1.3. Schenker's Work as Cognitive Psychology.

I view the greatest significance of Schenker's work to be its ability to examine in a concrete manner notions that previously were only felt intuitively by musicians. Indeed the analytic techniques and concepts articulated by Schenker are directly related to performance and compositional practices that stand at the center of the development of tonal music. An excellent example may be found in [FRAN78]. In this example it is made clear that the Ursatz of the composition fragment makes inevitable the final form of the composition. This syntactic

relationship between the Ursatz and the surface composition can be easily discerned aurally by the contemporary listener just as initially discovered aurally by Schenker.

In discovering the underlying structure of compositions Schenker opened the door to an understanding of a large part of the human process of composition. Allen Forte [FORT77] likens Schenker's achievement in music to that of Sigmund Freud's in psychology.

> Just as Freud opened the way for a deeper understanding of the human personality with his discovery that the diverse patterns of overt behavior are controlled by certain underlying factors, so Schenker opened the way for a deeper understanding of musical structure with his discovery that the manifold of surface events in a given composition is related in specific ways to a fundamental organization.

2.2. Previous Work.

Although the investigation of the computer generation of music based on grammars has been very limited, most if not all the approaches being applied to the study of natural language have also been applied to music. In his early work, Nicholas Ruwet [RUWE75] employed linguistic techniques to analyse works using formal grammars as a tool for verification of the analysis. By 1975, however, Ruwet suggested the use of a generative grammar as the correct approach. Arguing that intuitive construction of a theoretical model must precede testing of the model's validity by scientific procedures of validation or

invalidation. Ruwet by this time shifted the function of grammar from that of an inductive device for validation to a means for constructing deductive hypothetical theories (2).

Also writing in 1975, J-J Nattiez argued that significant synthetic (generative) rules could not be derived without a process of detailed "taxonomic" description. Using early procedures of Ruwet, Nattiez analysed scores of Stravinsky, Brahms and Varese to identify stylistic traits within "families" of similar scores, and suggested the construction of "syntagmatic grammars" to represent analysis of a particular work. Nattiez noted the analogy between Schenker's hierarchical graph notation and the tree structure of syntactic analysis he himself uses, but he criticized equating Schenker's Ursatz with Chomsky's semantic "deep structure" because he felt this led toward a "normative" conception of music. In later work, Nattiez argued against the use of grammatical models derived from analysis of extant work for compositional purposes on the grounds that such compositions are only "pastiches" of little independent value.

[LASK75] explored a "generative theory of music" based on Chomsky's schema [CHOM65] and regarded grammar as a form of theory. Arguing a sonological/psychological base, Laske adopted the use of regulated grammars to prescribe the order of tasks in

2. The work of Ruwet and the later work of Smoliar are both in clear agreement with the notion stated by Chomsky [CHOM65] that there are no procedures for discovering theories; there only exist methods of falsifying theories. Theories must be formulated intuitively.

the use of regulated grammars to prescribe the order of tasks in the compositional process. Ultimately, he studied musical composition as a project in cognitive psychology and incorporated music structures into a cognitive musicology, associating music structures with the behavior that produced them.

Lidov and Gabura [LIDO73] incorporated two grammars: a rhythmic base (a string of durations with indexes for tonal stress) and a pitch grammar (which interpreted stress indexes and assigned pitches to the rhythmic base) into a system whose goal was the generation of "common-practice" melodies similar to those found in Haydn's music and folk music. They concluded from their studies that a workable syntax was easier to achieve than a "convincing" contour.

Baroni and Jacoboni [BARO75], [BARO78] sought to define a grammar of melodies for certain Lutheran chorales. Derived from analysis of 60 chorales harmonized by J.S. Bach, this "grammar" of 56 rules was capable of producing "correct" (not aesthetic) melodies. Doubting the existence of a "deep musical structure", Baroni and Jacoboni felt that syntactic structures were not strictly correlated to semantic content and used a computer program to verify their generative hypotheses. In later work [BARO82], they used an "analogies tree" (transformational grammar that derived musical structure from a "kernel" through a series of transformations) to generate melodic forms.

Leonard Bernstein [BERN76] applied Chomsky's model of transformational grammar directly to tonal music. He used a rather simplistic set of analogies to describe the comparison (noun=melody, verb=rhythm, adjective=haarmony) and argued that the universals in music (the overtone series, tonic-dominant opposition) indicate an innate musical-grammatical competence, which reinforces Bernstein's belief in the "superiority" of tonal music over atonal .

Balaban [BALA81] constructed a "generalized concept model" representation for tonal music. Related to earlier linguistic theories, her model synthesizes theories of relational databases [CHEN76], attribute grammars [KNUT68] and semantic networks [WOOD75]. Her overall goal was to formalize basic tonal-music theory and to develop appropriate computer representations for this formalization. She studied only the parameters of pitch and duration, and developed formal representations of "tonal-music-strings," which are analogous to the "deep structures" of Chomsky's transformational generative grammar theories. She built her representation system on the notion of concepts (definable things, such as notes or chords), attributes (such as the number of notes in a chord) and the relationships among the concepts, and she formalized a Skeletal System in Lisp for simple formalization of conventional tonal music terminology.

Terry Winograd [WINO68] wrote a computer program to analyse the harmonic structures of various compositions using a "systemic

grammar" in which semantic routines are used to guide syntactic parsing. In his program, a preliminary syntactic analysis eliminates "context-sensitive" features and reduces the number of paths in the parsing. The program then does some initial parsing and assigns a degree of semantic "meaningfulness". In this way, parsing paths that are grammatical with respect to the codified harmony but not meaningful (i.e. part of the functional harmonic description of the piece) are eliminated. Winograd's system produced convincing results when applied to selections from Schubert and Bach. The principal of semantic parsing was used in the development of Winograd's natural language understanding system [WINO73], which has the capacity to "learn" from the environment both from externally fed and internally deduced information.

Curtis Roads [ROAD78] uses "composing grammars" in conjunction with a computer-run grammar interpreter as a means of organizing music structures. Roads designed two languages: TREE, a grammar specification language for music and COTREE, a composing language. Compositional works coded in COTREE are compiled into a score using grammar specified with TREE.

In summary, although a great many approaches to the computer generation of music have been attempted, none has been entirely successful. However the most promising approaches to the computer generation of music are grammatical approaches.

3. Implementation.

3.1. Overview

A composition may be viewed as a tree in which the leaves are the actual composition. The root is the tonal center of the composition. The branches connecting the root to the leaves form a parse tree of sorts for the composition. Consistent with this model, the system generates the outline for a musical composition through a series of transformations. The composition grows from the tonal center to complex surface structure as transformations are executed. The composition is stored as lists of notes. The first element of each list indicates the temporal relationships between all the items in the list. "Sim" indicates that all elements of the list are to occur simultaneously. "Seq" indicates that all elements of the list are to occur in sequence. Lists may be embedded within lists.

The system can be used interactively (in interpreted mode) or as a library of 'C' language functions that perform the transformations.

3.2. Program Modules.

In interpreted mode, the system parses the command line, performs the indicated transformations on the composition and displays the results. The user is then prompted and enters the next command line. Figure 3.1 displays the interaction of the modules when the system is used in interpreted mode. Initialization is performed in the Main() function. Next a loop is entered in which each command line is parsed. Command line parsing is performed by the

Parse() function. Parse() calls any of the transformation functions to actually perform the transformation that shapes the composition.

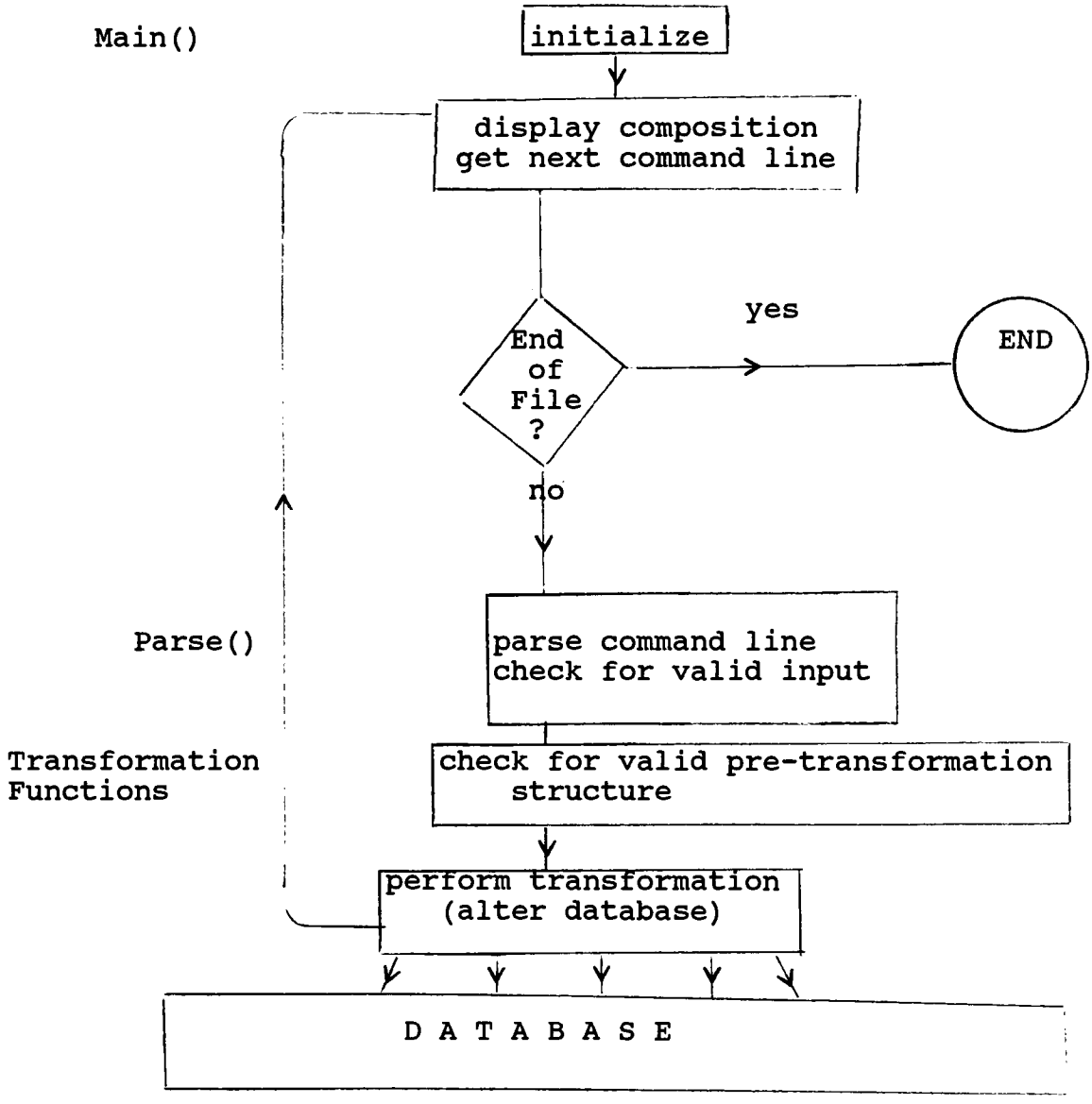


Fig. 3.1 System Modules

3.3. Implementation Functionality.

3.3.1 Data Structures

The system generates the outline for a composition through a set of transformations. This outline will consist of a set of notes and the temporal relationships between these notes. This means that although we know that certain notes or sets of notes should be played at the same time (a simultaneity) or one after the other (a sequence), no specific rhythm has been assigned to the playing of these notes.

A tree data structure is the basic data structure for a composition. Each tree node contains the following information:

1. Chromatic alteration - this field indicates any sharps or flats altering a single note event. In the following example, the integer 1 indicates that the note "A" is raised by 1/2 step (1 sharp added.)

   (1 A 0)

2. Event Name - each event is marked as a single note (A - G) or a simultaneity or sequence. "Sim" indicates a simultaneity. "Seq" indicates a sequence. The first element in a list of notes must be either "Sim" or "Seq". All elements of a list beginning with "Sim" are executed simultaneously. All elements of a list beginning with "Seq" are executed in sequence from left to right. In the following example, the first event name, "Seq" indicates that the second and third events, "A" and "B" the note A will be played followed by the note B.

   (Seq (0 A 0) (0 B 0))

3. Octave - middle C is the lowest note of octave 0. Octaves higher than this begin with Cn, where n > 0, and octaves lower begin with Cj, where j < 0 and each integer distance of 1 indicates 1 octave displacement.

(0 C 0) - middle C

4. Any additional events associated with a Simultaneity or Sequence event.

(Seq (0 A 0) (0 B 0))

Output is accomplished by a pre-order traversal of the tree. Each node, including all children and, recursively, each child's children is output enclosed in matching parentheses. Output is analagous to Lisp output in that each sub-composition is enclosed in matching parentheses like a Lisp list and in that the event "Sim" or "Seq" designating that the events in the list are a simultaneity or sequence, respectively, is always the first element of the list. Figure 3.2 shows an "extended" BNF description for a composition. An event is defined as either a note or a Simultaneity of events or a Sequence of events. A Simultaneity is defined as any number of events occurring at once. A sequence is defined as any number of events occurring one after the other. A note may be any diatonic note in the key of C major. Figure 3.3 is a sample compositional fragment consisting of the notes C -1, G -1 and C -1, each played one after the other. Figure 3.4 is the actual 'C' language source code for the tree node data structure (TNODE.)

```
event    ::=    note | Simultaneity | Sequence

                event1,

                event2,

Simultaneity    ::=    event3,
                          .
                          .
                          .

                eventn

              Time--------------------------------->

Sequence        ::=    event1, event2, event3 . . . eventn

              Time--------------------------------->

note            ::=    A | B | C | D | E | F | G
```

Fig. 3.2: "extended" BNF indicating time relevancy.

```
Example:  ( seq ( 0 C -1 ) ( 0 G -1 ) ( 0 C -1 ))
```

Fig. 3.3: compositional fragment

```
typedef struct tnode {
        char    note;
        int     octave;
        short   accidental;
        struct tnode    *child,
                        *brother;
}TNODE, *TNODEPTR;
```

Fig. 3.4: 'C' source code for TNODE data structure

Various "utility" functions are available. Input/Output functions are available, which allow the reading and writing

of compositions or sub-compositions. Macro processing functions also are available which allow the creation and use of command-line macros. The "focus" function allows the user to shift his current focus to a sub-composition. "Reset" restores the previous focus.

3.3.2 "Upper Level" Functions

The upper level functions are functions that shape an entire composition or sub-composition. These functions are nature, ursatz, binary and ternary. There is a mandatory order for the application of these functions as follows: The ternary function can only be applied on a composition that is the result of the binary transformation. The binary transformation can only be applied on a composition that is the result of a ursatz transformation and ursatz can only be applied on the result of the nature transformation. Each transformation can only transform a specific valid pre-transformation structured composition into its corresponding valid post-transformation structure. Therefore, there are a set of rules defining "what must happen first" before a transformation may be applied. The equivalence of a composition prior to a transformation and after any transformation is assumed. This includes the assumed equivalence of a single note with a corresponding "chord of nature." Therefore any set of transformations may be applied to any single note of a composition creating, recursively, an infinite number of compositional variations.

In usual usage, this system performs transformations, proceeding from a single note to a "chord of nature" consisting of a simultaneity of several notes of the natural overtone series above the single note. Next, the "ursatz" transformation is applied to the "chord of nature." At this point, the "binary" transformation and then the "ternary" transformation may be applied.

The first step in using this system must be to generate a "chord of nature" from a single note. In a new composition, this note is assumed to be the tonic of the current scale. This chord is comprised of the first 5, 6 or 7 tones of the natural overtone series and is the basis of any composition. This function requires a single parameter, 3, 5, or 8, which determines whether the upper triad is to be built to the third, fifth or octave respectively. This function may be abbreviated as "n". An example of the usage of the nature function is given in figure 3.5. In this example, the current scale is C.

```
==>  nature 3
( sim ( 0 C -2 )( 0 C -1 )( 0 G -1 )( 0 C 0 )( 0 E 0 ))
```

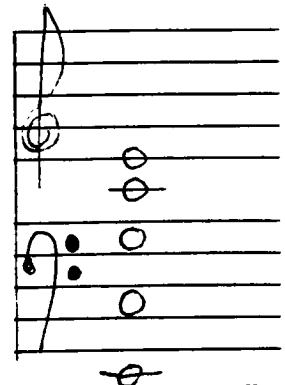


Fig. 3.5 Use of "Nature" function

The ursatz function must be performed on a composition or sub-composition in the form of a "chord of nature." This function transforms a chord of nature into its corresponding ursatz form. This function may be abbreviated to "u". An example of the usage of the ursatz function is given in figure 3.6.

```
abbreviation: u

==>  ursatz
( sim
   ( seq ( 0 E 0 )( 0 D 0 )( 0 C 0 ))
   ( seq ( 0 C -1 )( 0 G -1 )( 0 C -1 ))
)
```

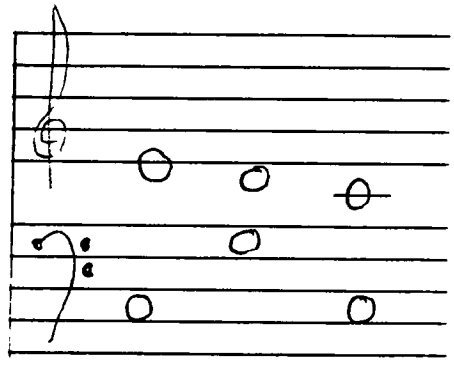


Fig. 3.6 Use of "Ursatz" function

BINARY

The binary function must be performed on a composition or sub-composition in the ursatz form. This function transforms a compositional entity in the ursatz form into its corresponding binary form. It takes no parameters. An example of the usage of the binary function is given in figure 3.7.

```
abbreviation: b

==>  binary
( seq
   ( sim
      ( seq ( 0 E 0 )( 0 D 0 ))
      ( seq ( 0 C -1 )( 0 G -1 ))
   )
   ( sim
      ( seq ( 0 E 0 )( 0 D 0 )( 0 C 0 ))
      ( seq ( 0 C -1 )( 0 G -1 )( 0 C -1 ))
   )
)
```

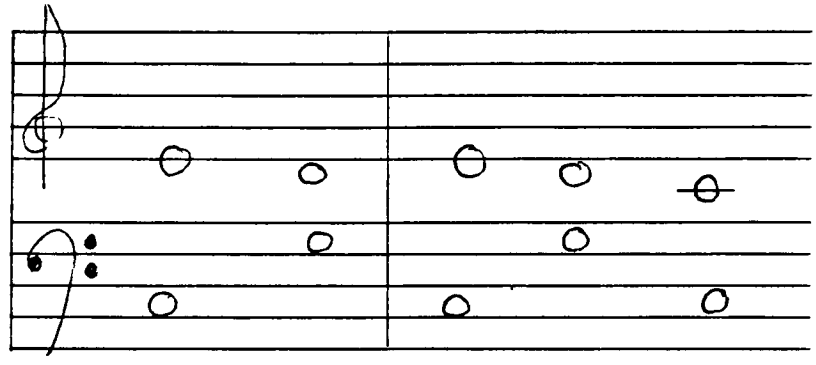


Fig. 3.7 Use of "Binary" function

TERNARY

The ternary function must be performed on a composition or sub-composition in the binary form. This function transforms a compositional entity in the binary form into its corresponding ternary form. It takes no parameters. An example of the usage of the ternary function is given in figure 3.8.

```
abbreviation: te

==>  ternary
( seq
   ( sim
      ( seq ( 0 E 0 )( 0 D 0 )( 0 C 0 ))
      ( seq ( 0 C -1 )( 0 G -1 )( 0 C -1 ))
   )
   ( sim ( 0 D 0 )( 0 G -1 ))
   ( sim
      ( seq ( 0 E 0 )( 0 D 0 )( 0 C 0 ))
      ( seq ( 0 C -1 )( 0 G -1 )( 0 C -1 ))
   )
)
```

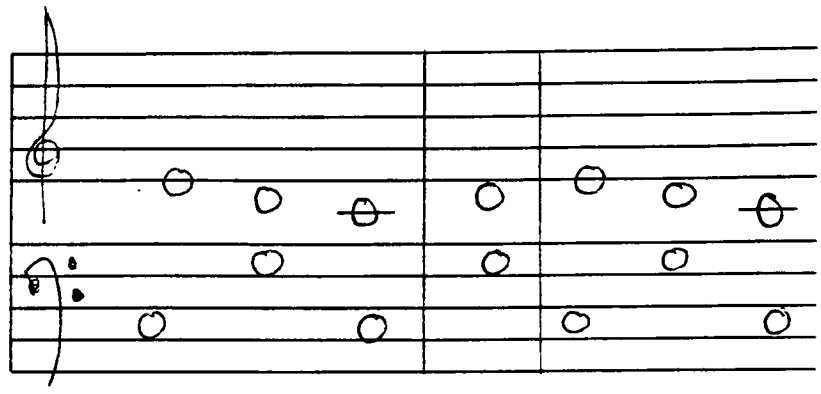


Fig. 3.8 Use of "Ternary" function

3.3.3 Lower Level Transformations

The lower level functions are functions that transform either single notes or sub-compositions that consist entirely of notes. These functions are: transpose, passingtone, auskomp, rauskomp, extend, ua, la, da, parallel, dim, sharp and flat. These functions may be applied in any order. The application of any lower level function is independent of the application of any other function. There is no ordered progression demanded, as there is in the application of upper level functions.

TRANSPOSE

usage: transpose [long] n

This function transforms a note or all notes in a sub-tree by transposing the note. If the optional argument "long" is used, each note in the current tree will be transposed. If the optional argument "long" is not used, a single note is transposed. The integer n indicates the number of octaves in the transposition. A positive integer indicates movement

upward, a negative integer indicates movement downward. An example of the usage of the transpose function is given in figure 3.9.

abbreviation: tr

(0 D 0)

==> transpose +2
(0 D 2)

(seq (0 C -1)(0 G -1)(0 C -1))

==> transpose long 1
(seq (0 C 0)(0 G 0)(0 C 0))

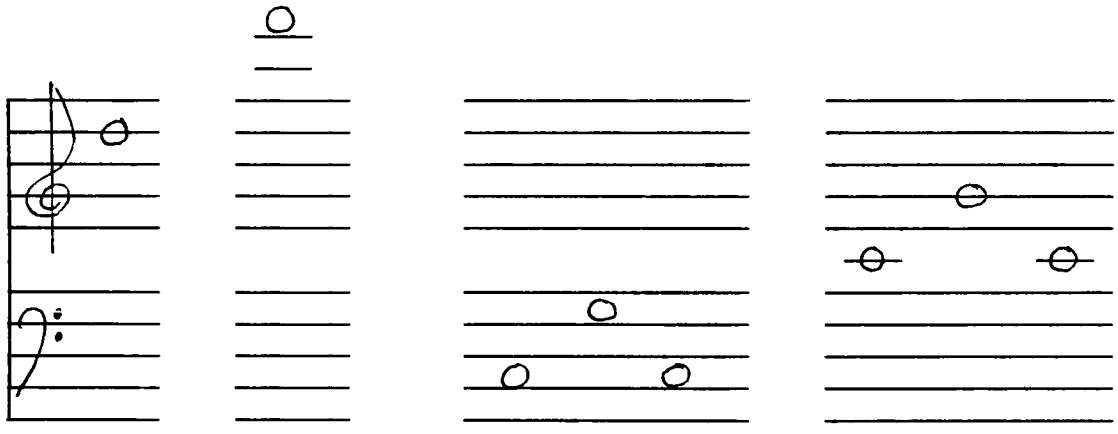


Fig. 3.9 Use of "Transpose" function

PASSINGTONE

This function transforms a sub-composition consisting of a sequence of notes. The function fills in diatonic notes between each pair of notes in the sequence which are not adjacent. It requires no parameters. An example of the usage of the passingtone function is given in figure 3.10.

```
abbreviation: pt

( seq ( 0 C 0 ) ( 0 G 0 ) ( 0 C 0 ) )

==>  passingtone
( seq
   ( 0 C 0 )
   ( seq ( 0 D 0 )( 0 E 0 )( 0 F 0 ))
   ( 0 G 0 )
   ( seq ( 0 F 0 )( 0 E 0 )( 0 D 0 ))
   ( 0 C 0 )
)
```

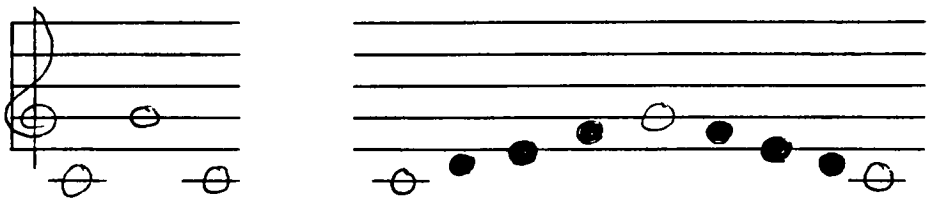


Fig. 3.10 Use of "Passingtone" function

AUSKOMP

This function transforms a sub-composition consisting of a simultaneity of notes into an ascending sequence of those notes. The function's name was coined by Smoliar and is from the German for "composing -out" (AUSKOMPonierung). It requires no parameters. An example of the usage of the auskomp function is given in figure 3.11.

```
abbreviation: a

( sim ( 0 D 0 )( 0 G -1 ))

==>  auskomp
( seq ( 0 G -1 )( 0 D 0 ))
```

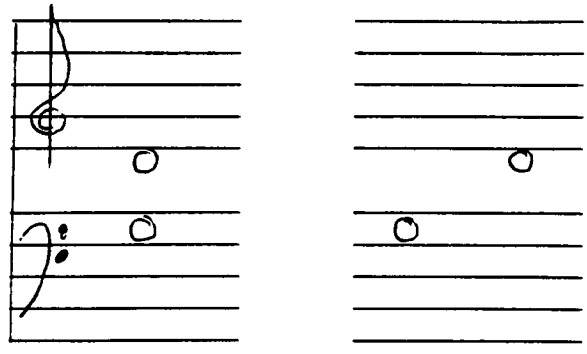


Fig. 3.11 Use of "Auskomp" function

RAUSKOMP

This function transforms a sub-composition consisting of a simultaneity of notes into a descending sequence of those notes. The function's name was coined by Smoliar with the intended implication that the function is the reverse of Auskomp, thus Rauskomp. This function requires no parameters. An example of the usage of the Rauskomp function is given in figure 3.12.

abbreviation: ra

```
( sim ( 0 D 0 )( 0 G -1 ))
==>  ra
( seq ( 0 D 0 )( 0 G -1 ))
```

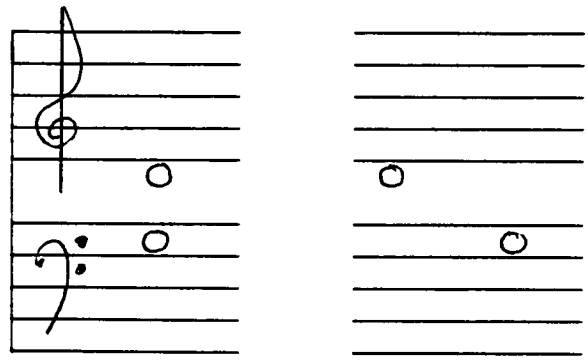


Fig. 3.12 Use of "Rauskomp" function

EXTEND

This function transforms a single note into a sequence of copies of that note. It takes a single parameter specifying the number of copies to create as the result of the transformation. An example of the usage of the Extend function is given in figure 3.13.

```
abbreviation: e
( 0 D 1 )

==>  extend 4
( seq ( 0 D 1 )( 0 D 1 )( 0 D 1 )( 0 D 1 ))
```

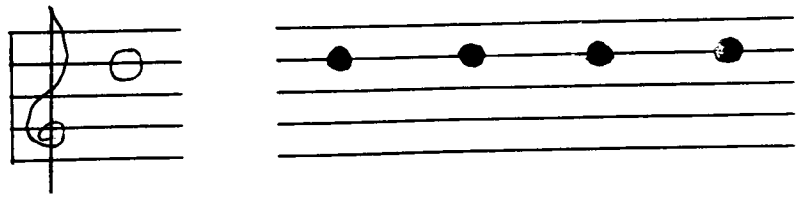


Fig. 3.13 Use of "Extend" function

UA

The ua or upper auxiliary function transforms a sub-composition consisting of a sequence of notes. The transformation consists of placing an Upper Auxiliary between any two consecutive notes which are identical. It requires no parameters. An example of the usage of the Ua function is given in figure 3.14.

```
abbreviation: ua

( seq ( 0 D 1 )( 0 D 1 )( 0 D 1 )( 0 D 1 ))

==>  ua
( seq
   ( 0 D 1 )
   ( seq ( 0 E 1 ))
   ( 0 D 1 )
   ( seq ( 0 E 1 ))
   ( 0 D 1 )
   ( seq ( 0 E 1 ))
   ( 0 D 1 )
)
```

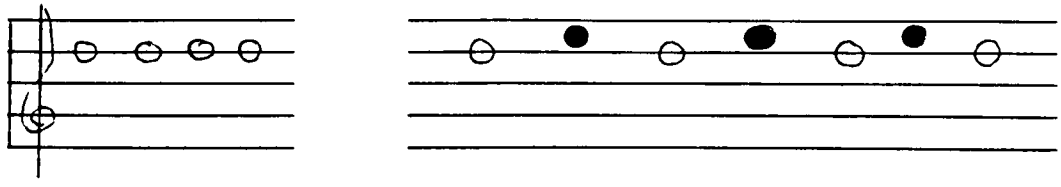


Fig. 3.14 Use of "Ua" function

LA

The la or lower auxiliary function transforms a sub-composition consisting of a sequence of notes. The transformation consists of placing a Lower Auxiliary between any two consecutive notes which are identical. It requires no parameters. An example of the usage of the "la" function is given in figure 3.15.

```
( seq ( 0 D 1 )( 0 D 1 )( 0 D 1 )( 0 D 1 ))

==>  la
( seq
   ( 0 E 1 )
   ( seq ( 0 D 1 ))
   ( 0 E 1 )
   ( seq ( 0 D 1 ))
   ( 0 E 1 )
   ( seq ( 0 D 1 ))
   ( 0 E 1 )
)
```

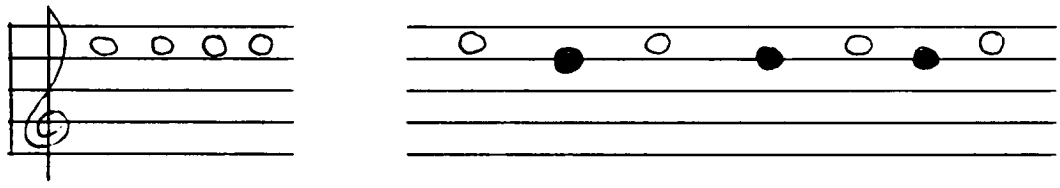


Fig. 3.15 Use of "La" function

DA

The da or double auxiliary function transforms a sub-composition consisting of a sequence of notes. The transformation consists of placing an upper and lower auxiliary between any two consecutive notes which are identical. It requires no parameters. An example of the usage of the "da" function is given in figure 3.16.

```
( seq ( 0 E 1 )( 0 E 1 )( 0 E 1 )( 0 E 1 )( 0 E 1 ))

==>  da
( seq
   ( 0 E 1 )
   ( seq ( 0 F 1 )( 0 D 1 ))
   ( 0 E 1 )
   ( seq ( 0 F 1 )( 0 D 1 ))
   ( 0 E 1 )
   ( seq ( 0 F 1 )( 0 D 1 ))
   ( 0 E 1 )
   ( seq ( 0 F 1 )( 0 D 1 ))
   ( 0 E 1 )
)
```

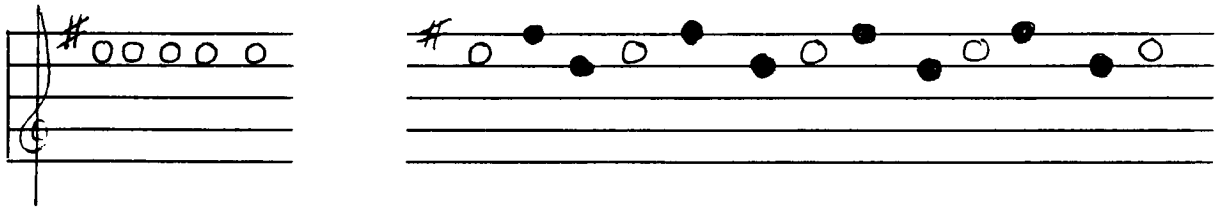


Fig. 3.16 Use of "Da" function

PARALLEL

The parallel function transforms a sub-composition consisting of a sequence of notes. This function adds one note for each note in the sequence. It takes one parameter, an integer specifying the interval distance between the notes of the existing sequence and the new "parallel" notes which will be added. An example of the usage of the "parallel" function is given in figure 3.17.

abbreviation: p

```
( seq ( 0 G 0 )( 0 F 0 )( 0 E 0 )( 0 D 0 )( 0 C 0 ))

==>  parallel 3
( seq
   ( sim ( 0 G 0 )( 0 B 0 ))
   ( sim ( 0 F 0 )( 0 A 0 ))
   ( sim ( 0 E 0 )( 0 G 0 ))
   ( sim ( 0 D 0 )( 0 F 0 ))
   ( sim ( 0 C 0 )( 0 E 0 ))
)
```

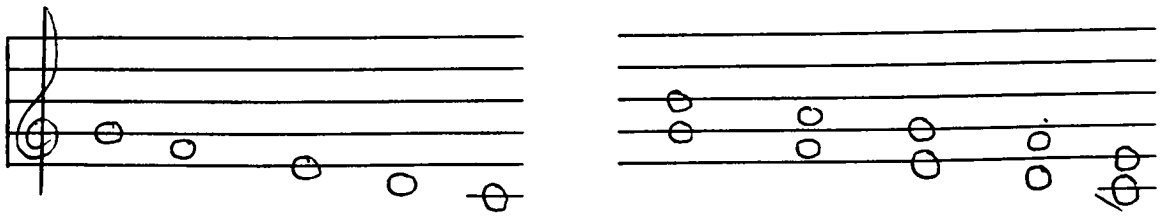


Fig. 3.17 Use of "parallel" function

DIM

The dim or diminution function transforms a sub-composition consisting of a sequence of notes. This function takes a single parameter, the positive integer n. It transforms the sequence of notes by replacing the first note in the sequence by a sequence consisting of the first n elements of the original sequence. An example of the usage of the "dim" function is given in figure 3.18.

abbreviation: di

```
( seq ( 0 G 0 )( 0 F 0 )( 0 E 0 )( 0 D 0 )( 0 C 0 ))

==>  dim 2
( seq
   ( seq ( 0 G 0 )( 0 F 0 ))
   ( 0 F 0 )
   ( 0 E 0 )
   ( 0 D 0 )
   ( 0 C 0 )
)
```

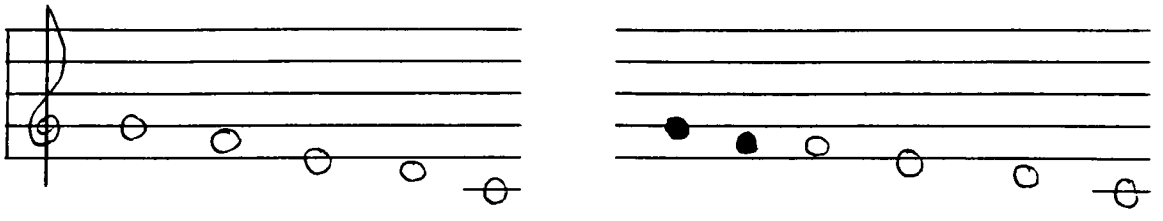


Fig. 3.18 Use of "dim" function

SHARP

This function transforms a single note by raising the note

1/2 tone. An example of the usage of the "sharp" function is given in figure 3.19.

abbreviation: sh

```
( 0 C 1 )

==>  sharp
( 1 C 1 )
```

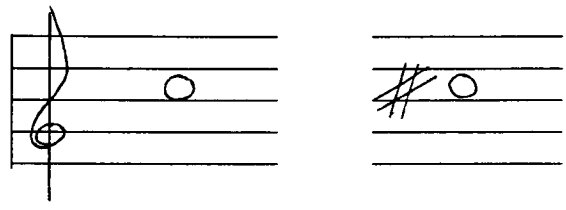


Fig. 3.19 Use of "sharp" funtion

FLAT

This function transforms a single note by lowering the note 1/2 tone. An example of the usage of the "flat" function is given in figure 3.20.

abbreviation: fl

```
( 0 B 1 )

==>  flat
( -1 B 1 )
```

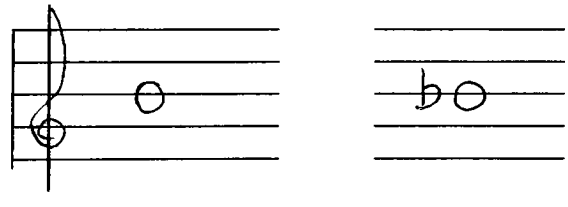


Fig. 3.20 Use of "flat" funtion

3.3.4 Utility Functions

The "utility" functions handle Input/Output, macro processing and maintain the user's "current focus" in a composition. These functions are: read, write, macro, macwrite, focus and reset.

READ, WRITE

The functions "Read" and "Write" allow the writing and reading of compositions or sub-compositions to and from disk files respectively. The syntax for these functions is as follows:

Read <input_file_name>

write <output_file_name>

MACRO

Macro is a macro processing function that allows the creation and use of command-line macros. The creation of a macro is started by typing "macro write <macro_name>" on the command line. The user follows this with the commands to be included in the macro. The creation of the macro is terminated by entering ".." on the command line. An example of the creation of the macro "mac1" is given if Fig. 3.21.

```
==> macro write mac1
==> nature 3
==> ursatz
==> binary
==> ..
```

Fig. 3.21 Creation of the macro "mac1"

The macro "mac1" now contains the three commands "nature 3", "urstaz" and "binary". It is invoked simply by entering "macro mac1" on the command line. An example of the use of the macro "mac1" is given in Fig. 3.22.

```
==> macro mac1
```

Fig. 3.22 Use of the macro "mac1"

The user may overwrite an existing macro by re-using the macro name when creating a new macro.

FOCUS, RESET

It is necessary to focus on a sub-composition in order to perform any lower-level transformation. It is also necessary to focus on a sub-composition if upper-level transformations will be performed recursively (i.e. when transforming a single note in an existing composition to a chord of nature.) The "focus" function allows the user to shift his current focus to a sub-composition. "Focus" addresses are stored in a push-down stack data structure. The syntax for the "focus" function is as follows:

```
focus n1 [ n2 ] [ n3 ] [ n4 ] ...
```

In the following example, the focus function will shift the current focus to the n4-th element of the n3-rd element of the n2-nd element of the n1-th sub-composition:

```
==> focus 3 5 2 6
```

Reset restores the current focus incrementally by "popping" "focus" addresses from a push-down stack of addresses. In the above example, the focus is restored in a single call to reset.

4. Results

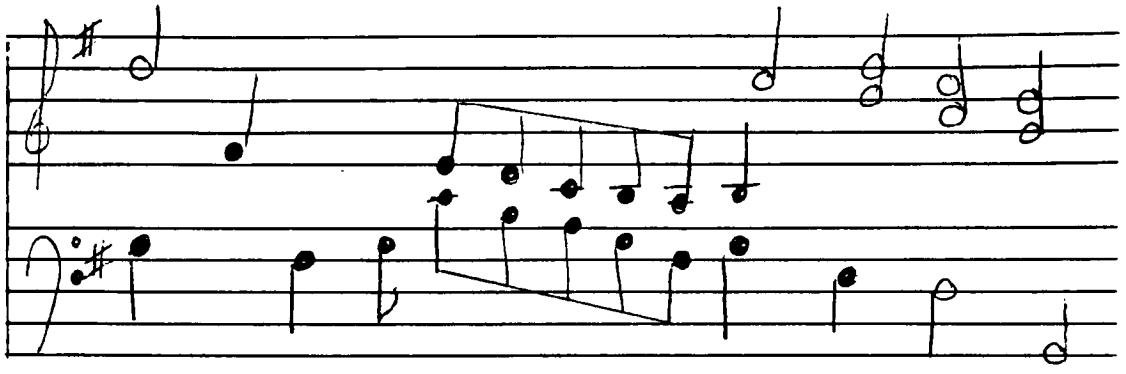
4.1 Results

This implementation is fully equivalent with the system defined by Smoliar [SMOL80]. I have tested all functions in the system by using the system in interactive mode. All functions perform as specified. The remainder of this chapter consists of excerpts from my replication of the analysis performed by Smoliar [SMOL80]. This is included in order to demonstrate the capabilities of the system and to display equivalence with Smoliar's definition of the system. The entire analysis is included in the appendix.

4.2 Sample Session

This sample dialogue constructs an analysis of the opening ten measures of Mozart's piano sonata, K. 283:

The purpose of the analysis will be to derive the following compositon skeleton:

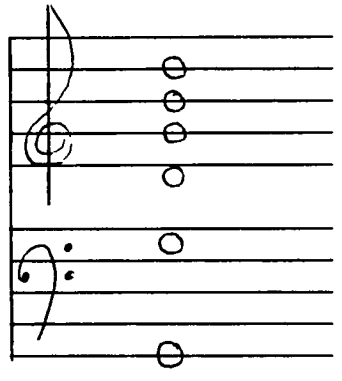


Scale sets the tonal center of the composition to "G".

==> scale g

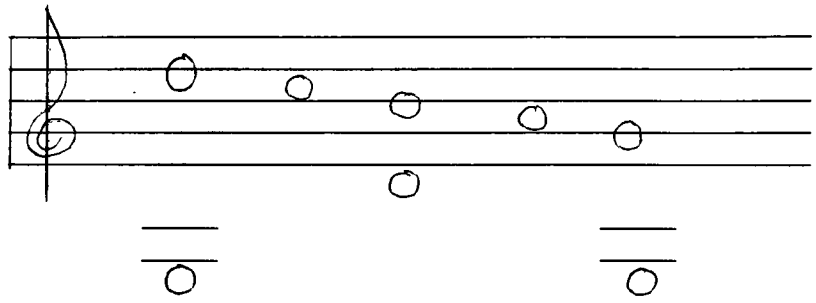
The first transformation performed is Nature. The notes of the natural overtone series for G are the basis for this composition.

==> nature 5
(sim (0 G -2)(0 G -1)(0 D 0)(0 G 0)(0 B 0)(0 D 1))



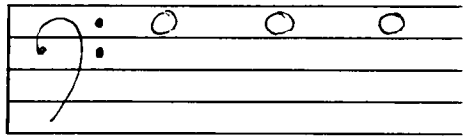
The basic shape for the composition is set by the Ursatz transformation.

```
==>  ursatz
( sim
   ( seq ( 0 D 1 )( 0 C 1 )( 0 B 0 )( 0 A 0 )( 0 G 0 ))
   ( seq ( 0 G -1 )( 0 D 0 )( 0 G -1 ))
)
```



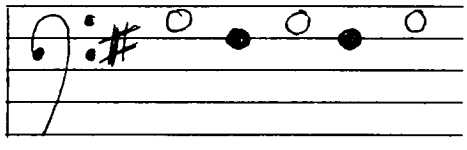
The bass notes are refined by inserting the note E -1 between the first and second notes, transposing the first bass note up one octave and extending it into three identical notes.

```
==>  focus 3 2
( 0 G -1 )
==>  extend 3
( seq ( 0 G -1 )( 0 G -1 )( 0 G -1 ))
```



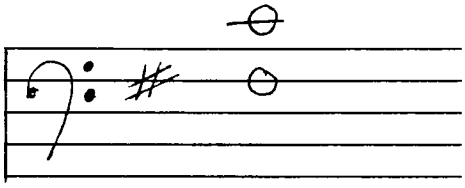
Lower auxiliaries are added:

```
==>  la
( seq
   ( 0 G -1 )
   ( seq ( 1 F -1 ))
   ( 0 G -1 )
   ( seq ( 1 F -1 ))
   ( 0 G -1 )
)
```



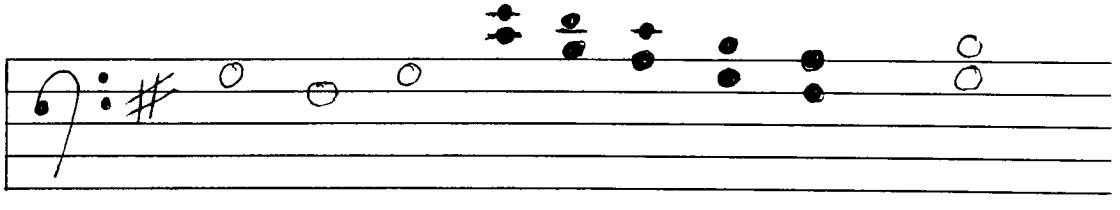
The last auxiliary is transformed by the addition of its parallel fifth.

```
==>  focus 5
( seq ( 1 F -1 ))
==>  parallel 5
( seq
   ( sim ( 1 F -1 )( 0 C 0 ))
)
==>  focus 2
( sim ( 1 F -1 )( 0 C 0 ))
```



These two notes are "composed out", passing tones are added and parallel notes are added at the interval of a third above the new descending notes.

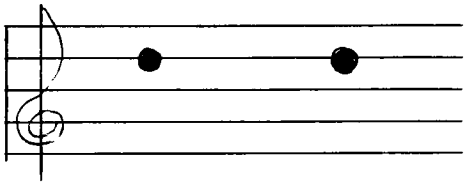
```
==>  focus 4 3
( 0 B -1 )
==>  remove
( seq
   ( sim ( 0 G -1 ))
   ( seq ( 1 F -1 ))
   ( sim ( 0 G -1 ))
   ( seq
      ( seq
         ( sim ( 0 C 0 )( 0 E 0 ))
         ( seq
            ( sim ( 0 B -1 )( 0 D 0 ))
            ( sim ( 0 A -1 )( 0 C 0 ))
            ( sim ( 0 G -1 )( 0 B -1 ))
         )
         ( sim ( 1 F -1 )( 0 A -1 ))
      )
   )
   ( sim ( 0 G -1 )( 0 B -1 ))
)
```



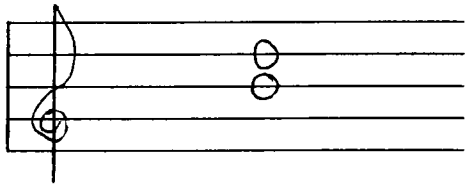
Minor additional transformations complete the bass.

The first treble note, D 1, is the only one transformed. It is extended into 2 identical notes. A parallel third is added. This is "composed-out" and parallel thirds are added to each of these notes.

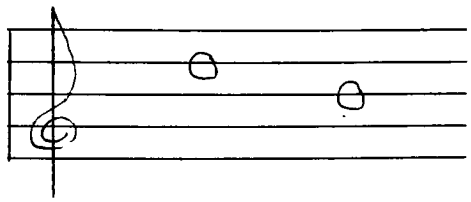
```
==>  focus 2
( seq ( 0 D 1 )( 0 C 1 )( 0 B 0 )( 0 A 0 )( 0 G 0 ))
==>  focus 2
( 0 D 1 )
==>  extend 2
( seq ( 0 D 1 )( 0 D 1 ))
```



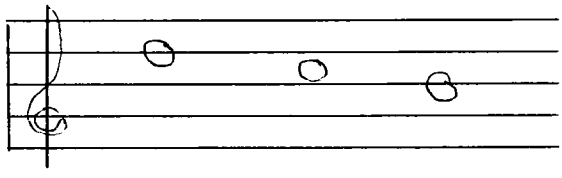
```
==>  parallel -3
( seq
   ( sim ( 0 D 1 )( 0 B 0 ))
   ( sim ( 0 D 1 )( 0 B 0 ))
)
==>  focus 3 2
( 0 D 1 )
==>  remove
( seq
   ( sim ( 0 D 1 )( 0 B 0 ))
   ( sim ( 0 B 0 ))
)
==>  focus 3
( sim ( 0 B 0 ))
==>  remove
( seq
   ( sim ( 0 D 1 )( 0 B 0 ))
)
==>  focus 2
( sim ( 0 D 1 )( 0 B 0 ))
```



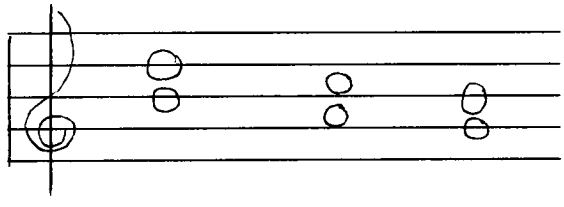
```
==>  rauskomp
( seq ( 0 D 1 )( 0 B 0 ))
```



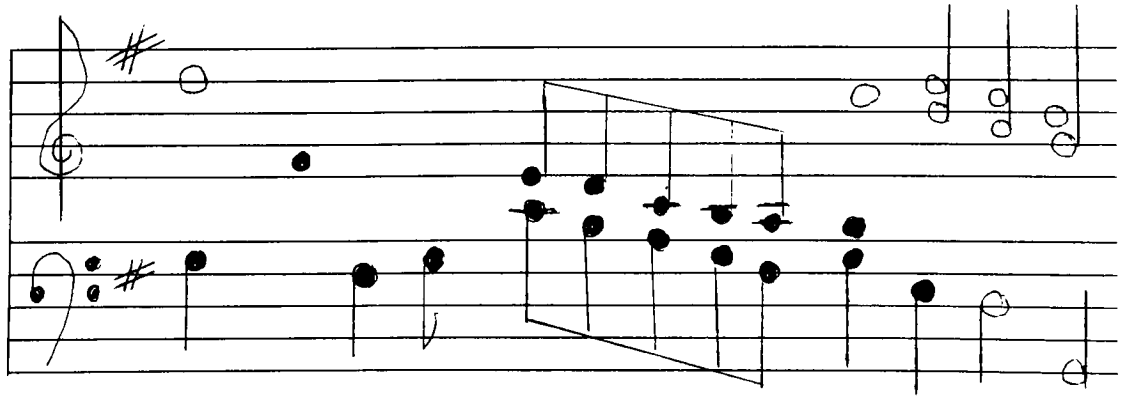
```
==>  passingtone
( seq
    ( 0 D 1 )
    ( seq ( 0 C 1 ))
    ( 0 B 0 )
)
==>  equalize
( seq ( 0 D 1 )( 0 C 1 )( 0 B 0 ))
```



```
==>  parallel -3
( seq
    ( sim ( 0 D 1 )( 0 B 0 ))
    ( sim ( 0 C 1 )( 0 A 0 ))
    ( sim ( 0 B 0 )( 0 G 0 ))
)
```



Minor additional changes complete the treble.



The resulting composition skeleton is equivalent with the analysis derived by Smoliar [SMOL80]. This proves the system's analytic equivalence with Smoliar's system. This analysis can be viewed as a functional test for the system because all major transformations are used. Additionally, this analysis proves the generative capability of the system because the analysis has been performed through generation.

5. Conclusions

Schenker's approach to the study of music theory is an extremely interesting way to examine tonal musical compositions. The set of transformations Smoliar derived from Schenker's work are a valid summary of Schenker's musical theories. These transformations comprise a grammar sufficiently robust to analyse compositions of the period from Bach to Brahms, with exceptions and additions as detailed in Section 2.1.1. My work is a full implementation of Smoliar's system. This system presently can be used as a computer aid in the performance of Schenkerian analysis as described by Smoliar [SMOL80].

There are several additional potential uses for my system implementation. One of these is in arranging music. The system can be used interractively to point toward or suggest proper harmonic structure for a melody. This is accomplished by analysing the melody and producing surface structure that is not in the melody and which can be read as suggestions for harmonic content.

The most promising potential use is the generation of compositions. To accomplish this by computer it will be necessary to combine the system with other AI systems that will determine the selection and sequence of transformations performed. The determination of which transformations will be performed and the sequence of the application of the transformations is beyond the capability of my system. My system will function as the "generative engine," in a

compositional system, producing valid transformations. These will be the basis for the composition.

The selection and ordering of transformations must be guided to generate valid compositions. An analogy can be drawn with the application of transformational grammars to natural language sentences. The following three individual transformations are among the transformations necessary to transform the sentence "Sylvester saw Tom" into the equivalent sentence "Tom was seen by Sylvester." :

Sylvester noun	is transformed to:	by Sylvester prepositional phrase in verb phrase
saw verb	is transformed to:	was seen passive verb
Tom verb phrase noun	is transformed to:	Tom noun phrase noun

Fig. 5.1 Sample Transformations

If any of the transformations is performed but all transformations are not performed, the result is an incorrect sentence. In the sentence "Saw Tom by Sylvester" the noun "Sylvester" has been transformed into the prepositional phrase in the verb phrase "by Sylvester". This is a valid transformation. However, the sentence resulting from this single transformation is incorrect.

Rules can be made that identify incorrect sentences. For example, a rule stating that each correct sentence must have

a noun will identify "Saw Tom by Sylvester" as an incorrect sentence.

Similarly, the correctness of compositions must be guaranteed. Perhaps rules can be defined stating the deep structure necessary for a valid composition. I feel this area should be investigated, but I do not feel this is the most promising area for investigation because this will reduce the generation of compositions to trial and error. This is not efficient for the generation of compositions. We should not focus on determing the correctness of existing compositions, even those which were just generated, but on the techniques of generating good compositions.

Several possible methods exist for determining the selection and sequence of transformations for the generation of a composition. One method is choosing transformations based on statistical observations of good compositions. There are obvious disadvantages to this approach. Analyses are not unique. Therefore, the statistical analyses that form the foundation of this method will be less than optimally informative. A more promising method of determining the selections and sequence of transformations will be defining a meta-language above the transformational grammar. Syntax rules for the meta-language will be derived from observing the deep structure of many correct compositions. Many Schenkerian analyses of good compositions must be performed as the basis for this meta-

language specification.

There are various possible extensions to this project. The "C" language interface, the toolkit, increases the system's potential as a generative system by allowing the system to be integrated easily into a larger AI system. Systems expert in the areas of rhythm, melody, contrapunctal movement and chord progression could be developed. Additional extensions would be the production of systems based on different sets of transformations, or systems that are the result of different grammar definitions. Possible examples are a system for the generation of atonal music and a system for the generation of twentieth century tonal music.

Because this system is a full implementation of Smoliar's system, research projects could be undertaken using this system consisting of the analysis of a large number of compositions. The goal of this research will be to develop statistical data on the relative frequency of appearance of the various transformations in existing compositions and the relative placement of these transformations to develop a statistically based procedure for the generation of compositions. An alternative to this statistical analysis would be the analysis of existing compositions to define a "meta-grammar", as described previously, for the generation of compositions.

In summary, I feel the application of grammatical approaches to the generation of all types of music is a very

interesting and promising area. The ultimate goal in this area should be automated generation of complete, interesting compositions. Although much more work needs to be done to succeed in computer generation of complete, interesting compositions, my project can be incorporated into a system that may achieve this goal.

BIBLIOGRAPHY

[APEL44]
Apel, Willi. Harvard Dictionary of Music (Cambridge MA: Harvard University Press, 1944)

[BALA81]
Balaban, M. 1981. Toward a Computerized Analytical Research of Tonal Music. Ph.D. thesis, Weizmann Institute of Science, Rehovat, Israel.

[BARO75]
Baroni, M. and C. Jacoboni. 1975. "Analysis and generation of Bach's chorale melodies." In G. Stefani, ed., Proceedings of the First International Congress on the Semiotics of Music. Pesaro, Italy: Centro di Iniziativa Culturale.

[BARO78]
Baroni, M., and C. Jacoboni. 1978 Proposal for a Grammar of Melody. Presses de l'Universite de Montreal. Bernstein, L. 1976. The Unanswered Question. Cambridge, Mass.: Harvard University Press.

[BARO82]
Baroni, M. 1982. "A Project of a Grammar of Melody." Presented at the First International Conference on Musical Grammars and Computer Analysis, Modena.

[BERN76]
Bernstein, L. 1976. The Unanswered Question. Cambridge, Mass.: Harvard University Press.

[CHEN76]
Chen, P. 1976 " The entity-relationship model - Toward a unified theory of data." ACM Transactions on Database Systems 1(1):9-36.

[CHOM65]
Chomsky, N. 1965. Aspects of the Theory of Syntax. Cambridge, Mass.: MIT Press.

[FORT77]
Forte, A. 1977. "Schenker's conception of music structure." Journal of Music Theory., Vol. 3, no.1 (4/1959).

[FRAN78]
Frankel, R.E.; Rosenschein, S.J.; and Smoliar, S.W. 1978. "Schenker's Theory of Tonal Music - Its Explication through Computational Processes." International Journal of Man-Machine Studies 10:121-138.

[KNUT68]
Knuth, D. 1968. "Semantics of context-free languages." Mathematical Systems Theory 2:127-145

[LASK75]
Laske, O. 1975. Introduction to a Generative Theory of Music. Sonological report 1b, Institute of Sonology, Utrecht.

[LIDO73]
Lidov, D., and J. Gabura. 1973. "A Melody writing algorithm using a formal language model." Computer Studies in the Humanities 4(3-4):138-148.

[MCHOSE]
McHose. "Normal Progression Chord Unification." Basic Principles of the Technique of 18th and 19th Century Composition

[OTTM61]
Ottman, R. 1961. Elementary Harmony Theory and Practice. Prentice Hall.

[ROAD85]
Roads, Curtis. 1985. "Music Grammars." Foundations of Computer Music ed. Roads, C. and Strawn, J. Cambridge, Mass.: MIT Press.

[ROAD78]
Roads, Curtis. 1978. "Composing Grammars," revised, Presented to the 1977 International Computer Music Conference, UCSD, La Jolla.

[RUWE75]
Ruwet,N. 1975. Theorie et Methodes dans les Etudes Musicales," Musique en Jeu, No. 17, Seuil, Paris.

[SCHE54]
Schenker, H. 1954. Harmony. ed. Jones, O., trans. Burgese, E. Chicago: Chicago University Press.

[SMOL80]
Smoliar, S. 1980. "A computer aid for Schenkerian analysis." Computer Music Journal 4(2):41-59.

[WINO68]
Winograd, T. 1968. "Linguistics and computer analysis of tonal harmony." Journal of Music Theory 12:2-49.

[WINO73]
Winograd, T. 1968. "A Procedural Model of Language Understanding," in Computer Models of Thought and Language, Shank, Colby (Eds.) W.H. Freeman, San Francisco. tonal harmony." Journal of Music Theory 12:2-49.

[WOOD75]
Woods, W. 1975 "What's in a link: Foundations for semantic networks." in D. Bobrow and A. Collins, eds., Representation and Understanding: Studies in Cognitive Science. New York: Academic.

7. Appendix

7.1. "C" Language Interface

DATA STRUCTURES

A tree is the basic data structure of all compositions. Each node of the tree corresponds to one note of the composition or the designation that the children of that node comprise a simultaneity or sequence.

```
typedef struct tnode {
        char    note;
        int     octave;
        short   accidental;
        struct tnode    *child,
                        *brother;
}TNODE, *TNODEPTR;
```

Examples:

When transformations produce new notes, these notes are diatonic in the current key. The current key can be defined as the values in the static "scales" table, which are indexed by "currentscale," each altered chromatically by the value of "chrom."

```
int     currentscale;
int     chrom;

struct scales {
        char    note;
        int     accidental;
};

extern struct scales    scale_tab[][7];
```

The address of the root of the tree containing the entire composition is stored in the variable "root." The address of the subtree currently being examined is stored in "cur_tree."

```
TNODEPTR        root;
TNODEPTR        cur_tree;
```

NAME

auskomp, binary, da, dim, equalize, exten, flat, isursatz, la, nature, note_transpose, Otranspose, parallel, pstone, rauskomp, sharp, ternary, ua, ursatz

SYNOPSIS

```
auskomp ( tree )
TNODEPTR tree;

binary ( tree )
TNODEPTR tree;

da ( tree )
TNODEPTR tree;

dim ( tree, n )
TNODEPTR tree;
int n;

equalize ( tree )
TNODEPTR tree;

extend ( n )
int n;

flat ( tree )
TNODEPTR tree;

isursatz ( tree )
TNODEPTR tree;

la ( tree )
TNODEPTR tree;

nature ( n )
int n;

note_transpose ( n )
int n;

Otranspose ( tree, n )
TNODEPTR tree;
int n;

parallel ( tree, n )
TNODEPTR tree;
int n;

pstone ( tree )
TNODEPTR tree;
```

```
rauskomp ( tree )
TNODEPTR tree;

sharp ( tree )
TNODEPTR tree;

ternary ( tree )
TNODEPTR tree;

ua ( tree )
TNODEPTR tree;

ursatz ( tree )
TNODEPTR tree;
```

DESCRIPTION

"auskomp" unfolds a simultaneity of notes into an ascending sequence.

"binary" transforms a tree which must be in the "ursatz" form to the appropriate "binary" form.

"da" (double auxiliary) inserts upper and lower diatonic auxiliaries between each identical pair of notes in a sequence of note.

"dim" (dimunition) transforms "tree" by replacing the first note in "tree" by a sequence of "n" notes consisting of the first "n" notes fo "tree. "Tree" must be a sequence of notes.

"equalize" flattens "tree". If any sequences or simultaneities exist within "tree" these sub-trees become incorporated into the next higher level of "tree."

"extend" transforms a single note into "n" identical occurances of that note.

"flat" lowers a single note by one-half step. It's argument must be a single note.

"isursatz" checks a tree to determine if it is in "ursatz" form.

"la" (lower auxiliary) inserts the lower diatonic auxiliary between each identical pair of notes in a sequence of note.

"nature" transforms the note at the address "cur_tree" (above) into a chord of nature. "Nature" takes a single argument, "n" which must be either 3 or 5 or 8. The top note of the new tree will be either the third, fifth or octave above the tonic, corresponding to the value of "n". "Tree" must be the tonic of the current scale.

"note_transpose" transposes "tree" "n" octaves. "N" may be either positive or negative. "Tree" must consist of a single note.

"Otranspose" transposes all "brothers" of "tree" a distance of "n" octaves.

"parallel" transforms "tree" by creating notes parallel to those in "tree" at an interval "n" above the notes in "tree." "Tree" must be a sequence of notes.

"pstone" fills in diatonic notes between each pair of notes which are not adjacent in a sequence of note.

"rauskomp" unfolds "tree", which must be a simultaneity of notes into a descending sequence of notes.

"sharp" raises a single note by one-half step. It's argument must be a single note.

"ternary" transforms a tree which must be in the "binary" form to the appropriate "ternary" form.

"ursatz" transforms a tree which must be in the "nature" form to the appropriate "ursatz" form.

"ua" (upper auxiliary) inserts the upper diatonic auxiliary between each identical pair of notes in a sequence of note.

NAME

rread - read in a composition from an external file

SYNOPSIS

```
TNODEPTR rread ( filename )
char *filename;
```

DESCRIPTION

"rread" reads a standard ASCII file and creates the tree defined by the contents of the ASCII file. The input file must describe a valid tree.

RETURN VALUE

Opon normal termination, "rread" returns a pointer to a tree containing the composition read. Null is returned if invalid input is encountered.

NAME

rremove - remove a tree

SYNOPSIS

```
rremove ( tree )
TNODEPTR tree;
```

DESCRIPTION

"rremove" removes "tree", including its sub-trees from the composition. Because this is accomplished by searching the entire composition starting at "root" (above), "root" must be defined.

NAME

tr_write - write a tree to standard output

SYNOPSIS

```
tr_write ( tree )
TNODEPTR tree;
```

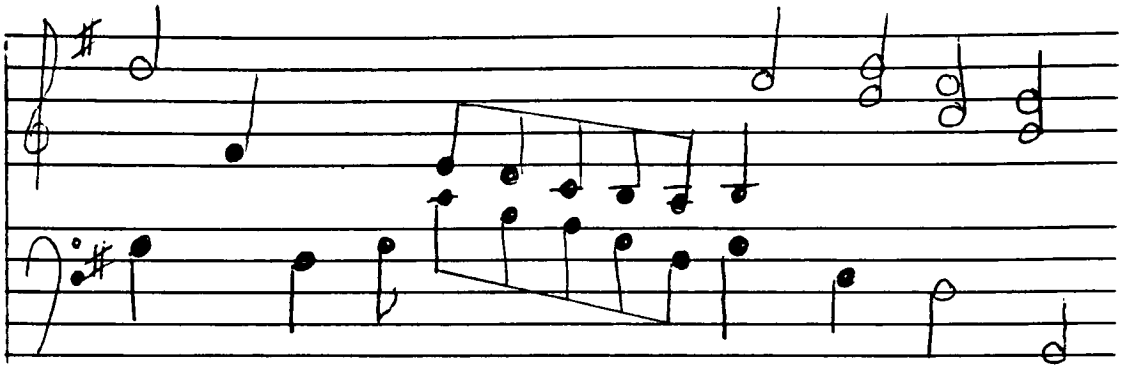
DESCRIPTION

"tr_write" writes the contents of "tree" to the standard output. Each sub-tree of "tree" is enclosed in parentheses. Each note of "tree" is displayed by the triplet accidental, note, octave enclosed in parentheses.

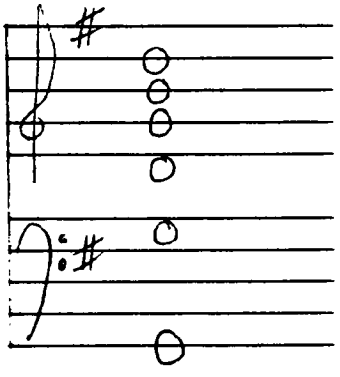
7.2. Sample Session

In the following sample dialogue I will construct a sample analysis of the opening ten measures of Mozart's piano sonata, K. 283:

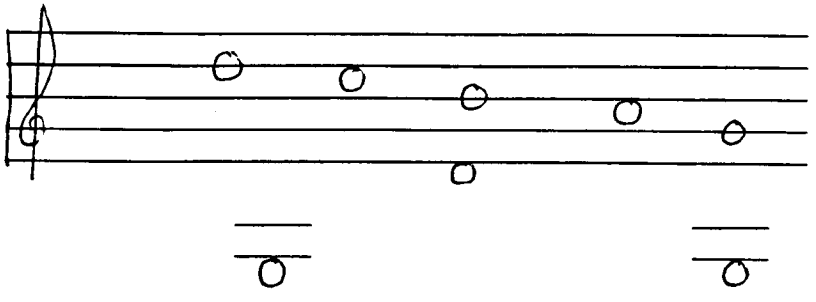
The purpose of the analysis will be to derive the following compositon skeleton:



```
==>  ==>  scale g
==>  nature 5
( sim ( 0 G -2 )( 0 G -1 )( 0 D 0 )( 0 G 0 )( 0 B 0 )( 0 D 1 ))
```

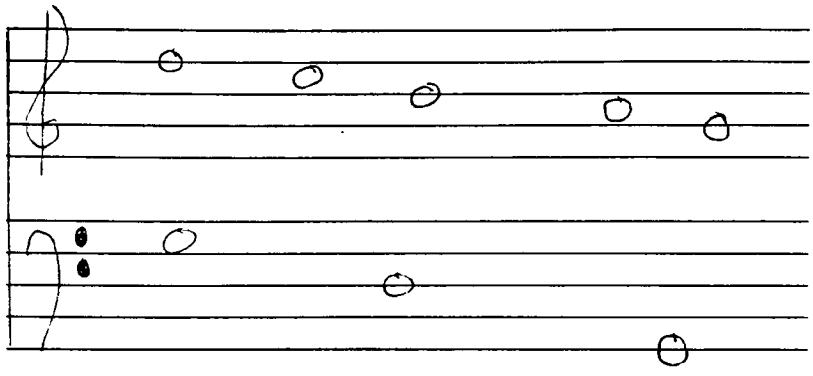


```
==>  ursatz
( sim
   ( seq ( 0 D 1 )( 0 C 1 )( 0 B 0 )( 0 A 0 )( 0 G 0 ))
   ( seq ( 0 G -1 )( 0 D 0 )( 0 G -1 ))
)
```



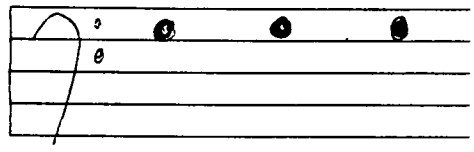
```
==>  focus 3
( seq ( 0 G -1 )( 0 D 0 )( 0 G -1 ))
==>  focus 4
( 0 G -1 )
==>  transpose -1
```

```
( 0 G -2 )
==>  reset
( seq ( 0 G -1 )( 0 D 0 )( 0 G -2 ))
```

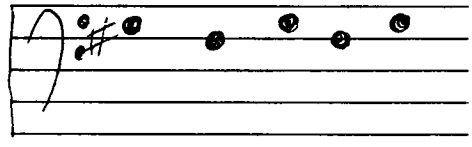


```
==>  focus 3
( 0 D 0 )
==>  extend 2
( seq ( 0 D 0 )( 0 D 0 ))
==>  ua
( seq
   ( 0 D 0 )
   ( seq ( 0 E 0 ))
   ( 0 D 0 )
)
==>  focus 2
( 0 D 0 )
==>  remove
( seq
   ( seq ( 0 E 0 ))
   ( 0 D 0 )
)
==>  reset
( seq
   ( 0 G -1 )
   ( seq
      ( seq ( 0 E 0 ))
      ( 0 D 0 )
   )
   ( 0 G -2 )
)
==>  reset
( sim
   ( seq ( 0 D 1 )( 0 C 1 )( 0 B 0 )( 0 A 0 )( 0 G 0 ))
   ( seq
      ( 0 G -1 )
      ( seq
         ( seq ( 0 E 0 ))
         ( 0 D 0 )
      )
      ( 0 G -2 )
   )
)
```

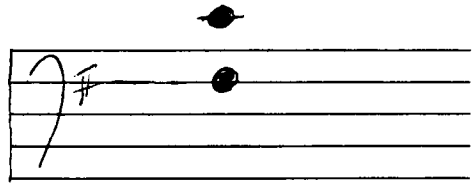
```
==>  focus 3 2
( 0 G -1 )
==>  extend 3
( seq ( 0 G -1 )( 0 G -1 )( 0 G -1 ))
```



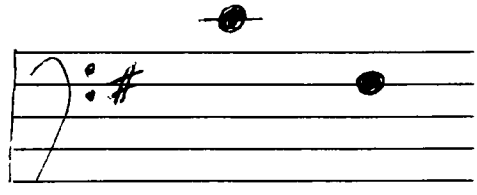
```
==>  la
( seq
    ( 0 G -1 )
    ( seq ( 1 F -1 ))
    ( 0 G -1 )
    ( seq ( 1 F -1 ))
    ( 0 G -1 )
)
```



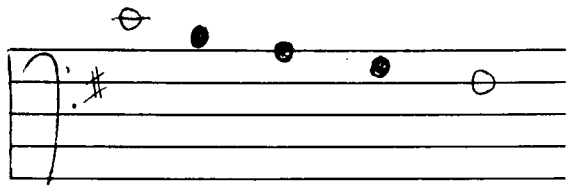
```
==>  focus 5
( seq ( 1 F -1 ))
==>  parallel 5
( seq
    ( sim ( 1 F -1 )( 0 C 0 ))
)
==>  focus 2
( sim ( 1 F -1 )( 0 C 0 ))
```



```
==>  rauskomp
( seq ( 0 C 0 )( 1 F -1 ))
```



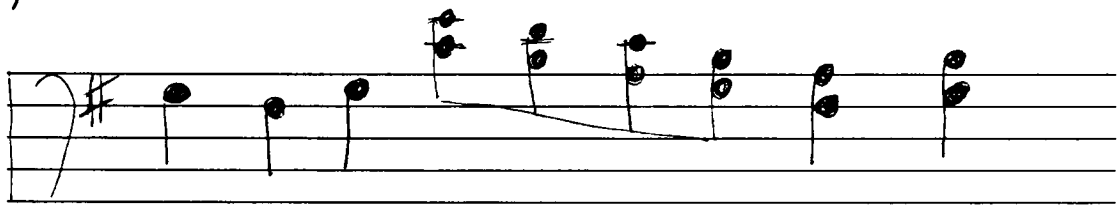
```
==>  passingtone
( seq
   ( 0 C 0 )
   ( seq ( 0 B -1 )( 0 A -1 )( 0 G -1 )( 0 D -1 ))
   ( 1 F -1 )
)
```



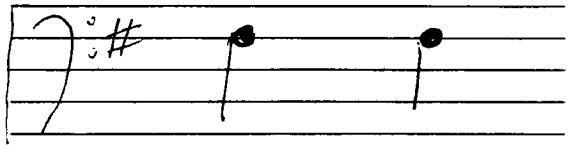
```
==>  parallel 3
( seq
   ( sim ( 0 C 0 )( 0 E 0 ))
   ( seq ( 0 B -1 )( 0 A -1 )( 0 G -1 )( 0 D -1 ))
   ( sim ( 1 F -1 )( 0 A -1 ))
)
==>  focus 3
( seq ( 0 B -1 )( 0 A -1 )( 0 G -1 )( 0 D -1 ))
==>  parallel 3
( seq
   ( sim ( 0 B -1 )( 0 D 0 ))
   ( sim ( 0 A -1 )( 0 C 0 ))
   ( sim ( 0 G -1 )( 0 B -1 ))
)
==>  reset
( seq
   ( sim ( 0 C 0 )( 0 E 0 ))
   ( seq
      ( sim ( 0 B -1 )( 0 D 0 ))
      ( sim ( 0 A -1 )( 0 C 0 ))
      ( sim ( 0 G -1 )( 0 B -1 ))
   )
   ( sim ( 1 F -1 )( 0 A -1 ))
)
==>  reset
( seq
   ( seq
      ( sim ( 0 C 0 )( 0 E 0 ))
      ( seq
         ( sim ( 0 B -1 )( 0 D 0 ))
         ( sim ( 0 A -1 )( 0 C 0 ))
         ( sim ( 0 G -1 )( 0 B -1 ))
      )
      ( sim ( 1 F -1 )( 0 A -1 ))
   )
)
```

```
==>  reset
( seq
   ( 0 G -1 )
   ( seq ( 1 F -1 ))
   ( 0 G -1 )
   ( seq
      ( seq
         ( sim ( 0 C 0 )( 0 E 0 ))
         ( seq
            ( sim ( 0 B -1 )( 0 D 0 ))
            ( sim ( 0 A -1 )( 0 C 0 ))
            ( sim ( 0 G -1 )( 0 B -1 ))
         )
         ( sim ( 1 F -1 )( 0 A -1 ))
      )
   )
   ( 0 G -1 )
)
==>  parallel 3
( seq
   ( sim ( 0 G -1 )( 0 B -1 ))
   ( seq ( 1 F -1 ))
   ( sim ( 0 G -1 )( 0 B -1 ))
   ( seq
      ( seq
         ( sim ( 0 C 0 )( 0 E 0 ))
         ( seq
            ( sim ( 0 B -1 )( 0 D 0 ))
            ( sim ( 0 A -1 )( 0 C 0 ))
            ( sim ( 0 G -1 )( 0 B -1 ))
         )
         ( sim ( 1 F -1 )( 0 A -1 ))
      )
   )
   ( sim ( 0 G -1 )( 0 B -1 ))
)
==>  focus 2 3
( 0 B -1 )
==>  remove
( seq
   ( sim ( 0 G -1 ))
   ( seq ( 1 F -1 ))
   ( sim ( 0 G -1 )( 0 B -1 ))
   ( seq
      ( seq
         ( sim ( 0 C 0 )( 0 E 0 ))
         ( seq
            ( sim ( 0 B -1 )( 0 D 0 ))
            ( sim ( 0 A -1 )( 0 C 0 ))
            ( sim ( 0 G -1 )( 0 B -1 ))
         )
         ( sim ( 1 F -1 )( 0 A -1 ))
      )
   )
   ( sim ( 0 G -1 )( 0 B -1 )) )
```

```
==>  focus 4 3
( 0 B -1 )
==>  remove
( seq
   ( sim ( 0 G -1 ))
   ( seq ( 1 F -1 ))
   ( sim ( 0 G -1 ))
   ( seq
      ( seq
         ( sim ( 0 C 0 )( 0 E 0 ))
         ( seq
            ( sim ( 0 B -1 )( 0 D 0 ))
            ( sim ( 0 A -1 )( 0 C 0 ))
            ( sim ( 0 G -1 )( 0 B -1 ))
         )
         ( sim ( 1 F -1 )( 0 A -1 ))
      )
   )
   ( sim ( 0 G -1 )( 0 B -1 ))
)
```

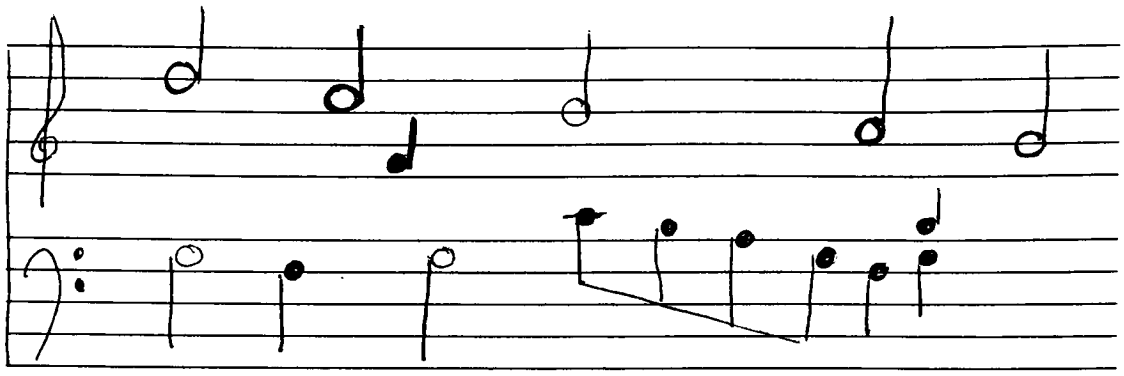


```
==>  focus 3 2
( 1 F -1 )
==>  extend 2
( seq ( 1 F -1 )( 1 F -1 ))
```



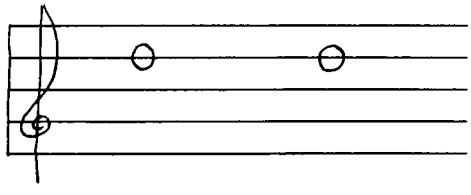
```
==>  focus 3
( 1 F -1 )
==>  transpose +1
( 1 F 0 )
==>  reset
( seq ( 1 F -1 )( 1 F 0 ))
```

```
==>  reset
( seq
   ( sim ( 0 G -1 ))
   ( seq
      ( seq ( 1 F -1 )( 1 F 0 ))
   )
   ( sim ( 0 G -1 ))
   ( seq
      ( seq
         ( sim ( 0 C 0 )( 0 E 0 ))
         ( seq
            ( sim ( 0 B -1 )( 0 D 0 ))
            ( sim ( 0 A -1 )( 0 C 0 ))
            ( sim ( 0 G -1 )( 0 B -1 ))
         )
         ( sim ( 1 F -1 )( 0 A -1 ))
      )
   )
   ( sim ( 0 G -1 )( 0 B -1 ))
)
==>  reset
( sim
   ( seq ( 0 D 1 )( 0 C 1 )( 0 B 0 )( 0 A 0 )( 0 G 0 ))
   ( seq
      ( seq
         ( sim ( 0 G -1 ))
         ( seq
            ( seq ( 1 F -1 )( 1 F 0 ))
         )
         ( sim ( 0 G -1 ))
         ( seq
            ( seq
               ( sim ( 0 C 0 )( 0 E 0 ))
               ( seq
                  ( sim ( 0 B -1 )( 0 D 0 ))
                  ( sim ( 0 A -1 )( 0 C 0 ))
                  ( sim ( 0 G -1 )( 0 B -1 ))
               )
               ( sim ( 1 F -1 )( 0 A -1 ))
            )
         )
         ( sim ( 0 G -1 )( 0 B -1 ))
      )
      ( seq
         ( seq ( 0 E 0 ))
         ( 0 D 0 )
      )
      ( 0 G -2 )
   )
)
```



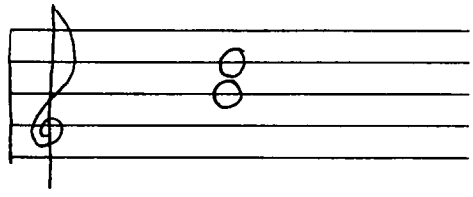
```
==>  reset
( sim
    ( seq ( 0 D 1 )( 0 C 1 )( 0 B 0 )( 0 A 0 )( 0 G 0 ))
    ( seq
       ( seq
          ( sim ( 0 G -1 ))
          ( seq
             ( seq ( 1 F -1 )( 1 F 0 ))
          )
          ( sim ( 0 G -1 ))
          ( seq
             ( seq
                ( sim ( 0 C 0 )( 0 E 0 ))
                ( seq
                   ( sim ( 0 B -1 )( 0 D 0 ))
                   ( sim ( 0 A -1 )( 0 C 0 ))
                   ( sim ( 0 G -1 )( 0 B -1 ))
                )
                ( sim ( 1 F -1 )( 0 A -1 ))
             )
          )
          ( sim ( 0 G -1 )( 0 B -1 ))
       )
       ( seq
          ( seq ( 0 E 0 ))
          ( 0 D 0 )
       )
       ( 0 G -2 )
    )
)
```

```
==>  reset
( sim
   ( seq ( 0 D 1 )( 0 C 1 )( 0 B 0 )( 0 A 0 )( 0 G 0 ))
   ( seq
      ( seq
         ( sim ( 0 G -1 ))
         ( seq
            ( seq ( 1 F -1 )( 1 F 0 ))
         )
         ( sim ( 0 G -1 ))
         ( seq
            ( seq
               ( sim ( 0 C 0 )( 0 E 0 ))
               ( seq
                  ( sim ( 0 B -1 )( 0 D 0 ))
                  ( sim ( 0 A -1 )( 0 C 0 ))
                  ( sim ( 0 G -1 )( 0 B -1 ))
               )
               ( sim ( 1 F -1 )( 0 A -1 ))
            )
         )
         ( sim ( 0 G -1 )( 0 B -1 ))
      )
      ( seq
         ( seq ( 0 E 0 ))
         ( 0 D 0 )
      )
      ( 0 G -2 )
   )
)
==>  focus 2
( seq ( 0 D 1 )( 0 C 1 )( 0 B 0 )( 0 A 0 )( 0 G 0 ))
==>  focus 2
( 0 D 1 )
==>  extend 2
( seq ( 0 D 1 )( 0 D 1 ))
```

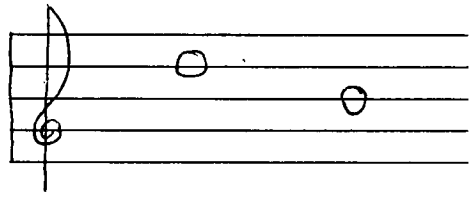


```
==>  parallel -3
( seq
   ( sim ( 0 D 1 )( 0 B 0 ))
   ( sim ( 0 D 1 )( 0 B 0 ))
)
==>  focus 3 2
( 0 D 1 )
```

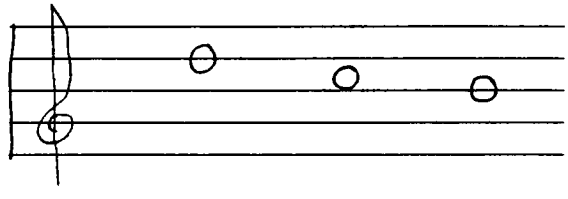
```
==>  remove
( seq
   ( sim ( 0 D 1 )( 0 B 0 ))
   ( sim ( 0 B 0 ))
)
==>  focus 3
( sim ( 0 B 0 ))
==>  remove
( seq
   ( sim ( 0 D 1 )( 0 B 0 ))
)
==>  focus 2
( sim ( 0 D 1 )( 0 B 0 ))
```



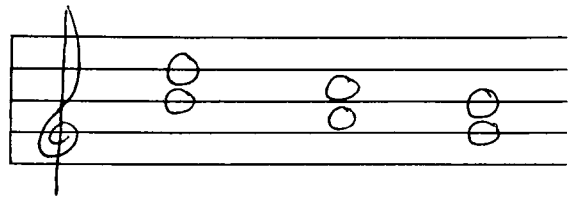
```
==>  rauskomp
( seq ( 0 D 1 )( 0 B 0 ))
```



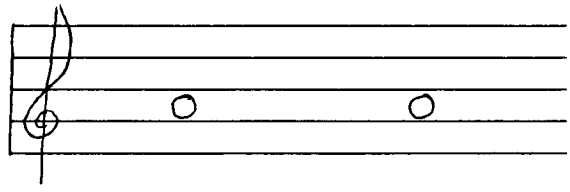
```
==>  passingtone
( seq
   ( 0 D 1 )
   ( seq ( 0 C 1 ))
   ( 0 B 0 )
)
==>  equalize
( seq ( 0 D 1 )( 0 C 1 )( 0 B 0 ))
```



```
==>  parallel -3
( seq
   ( sim ( 0 D 1 )( 0 B 0 ))
   ( sim ( 0 C 1 )( 0 A 0 ))
   ( sim ( 0 B 0 )( 0 G 0 ))
)
```



```
==>  focus 3 3
( 0 A 0 )
==>  extend 2
( seq ( 0 A 0 )( 0 A 0 ))
```



```
==>  focus 2
( 0 A 0 )
==>  transpose -1
( 0 A -1 )
==>  reset
( seq ( 0 A -1 )( 0 A 0 ))
==>  reset
( seq
   ( sim ( 0 D 1 )( 0 B 0 ))
   ( sim
      ( 0 C 1 )
      ( seq ( 0 A -1 )( 0 A 0 ))
   )
   ( sim ( 0 B 0 )( 0 G 0 ))
)
==>  reset
( seq
   ( seq
      ( sim ( 0 D 1 )( 0 B 0 ))
      ( sim
         ( 0 C 1 )
         ( seq ( 0 A -1 )( 0 A 0 ))
      )
      ( sim ( 0 B 0 )( 0 G 0 ))
   )
)
```

```
==>  reset
( seq
  .( seq
       ( seq
          ( sim ( 0 D 1 )( 0 B 0 ))
          ( sim
             ( 0 C 1 )
             ( seq ( 0 A -1 )( 0 A 0 ))
          )
          ( sim ( 0 B 0 )( 0 G 0 ))
       )
   )
   ( 0 C 1 )
   ( 0 B 0 )
   ( 0 A 0 )
   ( 0 G 0 )
)
```

```
==>  reset
( sim
   ( seq
      ( seq
         ( seq
            ( sim ( 0 D 1 )( 0 B 0 ))
            ( sim
               ( 0 C 1 )
               ( seq ( 0 A -1 )( 0 A 0 ))
            )
            ( sim ( 0 B 0 )( 0 G 0 ))
         )
      )
      ( 0 C 1 )
      ( 0 B 0 )
      ( 0 A 0 )
      ( 0 G 0 )
   )
   ( seq
      ( seq
         ( sim ( 0 G -1 ))
         ( seq
            ( seq ( 1 F -1 )( 1 F 0 ))
         )
         ( sim ( 0 G -1 ))
         ( seq
            ( seq
               ( sim ( 0 C 0 )( 0 E 0 ))
               ( seq
                  ( sim ( 0 B -1 )( 0 D 0 ))
                  ( sim ( 0 A -1 )( 0 C 0 ))
                  ( sim ( 0 G -1 )( 0 B -1 ))
               )
               ( sim ( 1 F -1 )( 0 A -1 ))
            )
         )
         ( sim ( 0 G -1 )( 0 B -1 ))
      )
      ( seq
         ( seq ( 0 E 0 ))
         ( 0 D 0 )
      )
      ( 0 G -2 )
   )
)
```

```
#  MAKEFILE
# *********************************

CFILES= extern.c func.c getword.c nature.c parse.c split.c test.c main.c \
        new.c overwrite.c write.c brother.c scale.c init.c child.c key.c \
        unfold.c free.c focus.c pstone.c isursatz.c fillin.c binary.c \
        checkdigit.c isbinary.c

OFILES1= extern.obj func.obj getword.obj nature.obj split.obj overwrite.obj \
        brother.obj scale.obj init.obj child.obj free.obj binary.obj

OFILES2= write.obj parse.obj new.obj main.obj key.obj unfold.obj oct_tran.obj\
        ursatz.obj focus.obj pstone.obj isursatz.obj test.obj fillin.obj \
        checkdigit.obj isbinary.obj


prog.exe:    $(OFILES1) $(OFILES2)
        link  $(OFILES1) $(OFILES2)  ,  prog.exe

$(OFILES1) :     def.h

$(OFILES2) :     extern.h def.h



/*************************************
def.h
*************************************/

#define FOREVER for(;;)

#define MAXARGS 64
#define BUFSIZE 128

#define TEST     0        /* magic number for functions   */
#define NATURE   1        /* must agree with index of functab */
#define SCALE    2
#define TRANSPOSE 3
#define URSATZ   4
#define FOCUS    5
#define RESET    6
#define PASSTONE 7
#define BINARY   8
#define CLEAR    9
#define HELP    10
#define TERNARY 11
#define EXTEND  12
#define AUSKOMP 13
#define RAUSKOMP 14
#define UA       15
#define LA       16
#define DA       17
#define SHARP    18
#define FLAT     19
#define PARALLEL 20
#define REMOV    21
#define EQUALIZE 22
#define DIM      23
#define WRIT     24
#define READ     25
#define MACRO    26
#define UNDO     27

#define TONIC            ( scale_tab[currentscale][0] )
#define SUPERTONIC       ( scale_tab[currentscale][1] )
```

```
#define MEDIANT          ( scale_tab[currentscale][2] )
#define SUBDOMINANT      ( scale_tab[currentscale][3] )
#define DOMINANT         ( scale_tab[currentscale][4] )
#define SUPERDOMINANT    ( scale_tab[currentscale][5] )
#define SUBTONIC         ( scale_tab[currentscale][6] )
#define SIM              'h'
#define SEQ              'i'

#define C       0
#define G       1
#define D       2
#define A       3
#define E       4
#define B       5
#define F       6

#define copy(p,q)    {(q->note)=(p->note);(q->accidental)=(p->accidental);\
                        (q->octave) = (p->octave)   ;}

#define fulcopy(p,q)  {(q->note)=(p->note);(q->accidental)=(p->accidental);\
                       (q->child) = (p->child);  (q->brother)=(p->brother);\
                        (q->octave) = (p->octave)   ;}


#define inc( A,B)    (  (((A) < 'C'  && (B) <  'C' && (A) > (B)) ||       \
                         ((A) < 'C'  && (B) >= 'C' && (A) < (B)) ||       \
                         ((A) >= 'C' && (B) >= 'C' && (A) > (B)) )        \
                          ? 1 : 0 )


#define dec( A,B)    (  (((A) < 'C'  && (B) <  'C' && (A) > (B)) ||       \
                         ((A) < 'C'  && (B) >= 'C' ) ||  \
                         ((A) >= 'C' && (B) >= 'C' && (A) > (B)) )        \
                          ? 0 : -1 )

#define dec2( A,B)    (  (((A) < 'C'  && (B) <  'C' && (A) >= (B)) ||      \
                         ((A) < 'C'  && (B) >= 'C' ) ||  \
                         ((A) >= 'C' && (B) >= 'C' && (A) >= (B)) )       \
                          ? 0 : -1 )


/*************************************
extern.h
*************************************/

#include        "def.h"

typedef struct tnode {
        char    note;
        int     octave;
        short   accidental;
        struct tnode    *child,
                        *brother;
}TNODE, *TNODEPTR;

extern  TNODEPTR        root;
extern  TNODEPTR        cur_tree;
extern  char    buf[BUFSIZE];

extern  char    *gargv[];
extern  int     gargc;

extern  int     currentscale;
extern  int     chrom;
```

```
        char    note;
        int     accidental;
};

extern struct scales    scale_tab[][7];

extern int      verbose;
extern char     *PS1;

extern TNODEPTR focuslist[128];
extern int      focusindex;

extern char     last[];
extern char     prev[];

/*********************************
auskomp.c
*********************************/

#include <stdio.h>
#include "extern.h"

#define SORTSIZE 64

auskomp(old)
TNODEPTR        old;

{

int     a,
        gap,
        i,
        j,
        n,
        tempkeynumber,
        num_items;

TNODEPTR        p,
                q,
                tempptr;

struct  {
        TNODEPTR        ptr;
        int       keynumber;
                } sorttab[SORTSIZE];


/*   check that all nodes in subtree are actually notes */

if ( old->note != SIM ){
        error ( "auskomp" , "must 'auskomp' a simultaneity of notes");
        return (0 );
}
for ( p = old->child; p != NULL; p = p->brother )
        if ( !(p->note >= 'A' && p->note <= 'G' ) ){
                error ( "auskomp" , "must 'auskomp' a simultaneity of notes");
                return (0 );
        }

/* checks out okay so proceed */

for ( p = old->child,i = 0; p != NULL; p = p->brother, i++ ){
        sorttab[i].ptr = p;
        sorttab[i].keynumber = num(p );
}
```

```
num_items = i;


for ( gap = num_items/2; gap > 0; gap /= 2 )
        for ( j = gap; j < num_items ; j++ )
                for ( i = j - gap; i >= 0; i -= gap ){
                        if ( sorttab[i].keynumber < sorttab[i+gap].keynumber )
                                break;
                        tempptr = sorttab[i].ptr;           /* swap values */
                        tempkeynumber = sorttab[i].keynumber;
                        sorttab[i].keynumber = sorttab[i+gap].keynumber;
                        sorttab[i].ptr = sorttab[i+gap].ptr;
                        sorttab[i+gap].ptr = tempptr;
                        sorttab[i+gap].keynumber = tempkeynumber;
                }



for ( i = 0; i < (num_items-1); i++ ){
        p = sorttab[i].ptr;
        p->brother = sorttab[i+1].ptr;
}

sorttab[num_items-1].ptr->brother = NULL;

old->note = SEQ;
old->child = sorttab[0].ptr;

}


/*************************************
binary.c
*************************************/


/*  The function's argument is a TNODEPTR.  The function transform a chord
     in the ursatz form into the corresponding binary form.  */


#include        <stdio.h>
#include        "extern.h"

TNODEPTR        binary( old )
TNODEPTR        old;
{
TNODEPTR        p,
                q,
                r,
                s,
                temp,
                new();

if ( ! ( isursatz ( old ) ) )
        return ( 0 );


p = old->child->child;                  /* copy treble notes    */
s = q = new();
copy ( p, q);

p = p-> brother;
q = q->brother = new();

while ( p->note != TONIC.note ){                /* loop down to supertonic */
```

```
        if ( p->brother->note != TONIC.note )
                q = q->brother = new();
        p = p-> brother;
}


p = old->child->brother->child; /* copy bass notes      */
r = q = new();
copy ( p, q);

p = p-> brother;
q = q->brother = new();
copy ( p, q);



p = new();              /* assign new hierachy on top of ursatz form. */
p->note = SEQ;
q = p->child = new();
q->note = SIM;

temp = new();
fulcopy(old,temp);
q->brother = temp;

fulcopy(p,old);

/*old = p;*/

q = q->child = new();
q->note = SEQ;
q->child = s;           /* treble       */
q = q->brother = new();
q->note = SEQ;
q->child = r;           /* bass         */

return (old);
}

/*************************************
brother.c
*************************************/

#include        "extern.h"
#include        "def.h"
#include        <stdio.h>
#include        <ctype.h>

TNODEPTR        brother( step, p )
int step;
TNODEPTR        p;
{

while ( --step >= 0 )
        if ( p->brother )
                p = p->brother;
        else{
                error( gargv[0], "not enough notes in list" );
                return ( ( TNODEPTR ) 0 );
        }
return ( p );
}


/*************************************
```

```
*********/

#include "extern.h"
#include <stdio.h>
#include <ctype.h>                  /* check each string from gargv[n] to last
                                    command line argument to ensure that they
                                    are all digits  */

checkdigit ( s, n )

char    *s;
int     n;

{

int     i;

while ( n < gargc ){
        for (i=0; isdigit (gargv[n][i]) && gargv[n][i] != '\0'; i++);

        if (  gargv[n][i] == '\0' )
                n++;
        if (  gargv[n][i] != '\0' )
                break;
}

if ( n == gargc )
        return ( 1 );
else{
        execerror (s, "non-numeric argument:", gargv[n]);
        return ( NULL );
}
}


/*************************************
child.c
*************************************/

#include        "extern.h"

TNODEPTR        child( step, p )
int step;
TNODEPTR        p;
{

while ( --step >= 0 )
        if ( p->child )
                p = p->child;
        else{
                error( gargv[0], "not enough notes in list" );
                return ( ( TNODEPTR ) 0 );
        }
return ( p );
}


/*************************
da.c
*************************/

#include <stdio.h>
#include "extern.h"

da (old )
TNODEPTR        old;
```

```
TNODEPTR        p,
                q,
                r,
                s,
                new();
int     i;

if ( old->note != SEQ || old->child == NULL ){
        error ( "double auxiliary", "this tree is not in the proper form" );
        return ( 0 );
}

for ( q = old->child, p = q->brother; p != NULL; q = p, p = p->brother )
        if ( p->note == q->note && p->octave == q->octave &&
                                                p->note >= 'A'
                                                &&  p->note <= 'G' ){
                        s = new();
                        i = (findindex(currentscale,p->note)+1)%7;
                        s->note = scale_tab[currentscale][i].note;
                        s->accidental = scale_tab[currentscale][i].accidental
                                                                + chrom;
                        if ( p->note == 'B' )
                                s->octave = p->octave +1;
                        else
                                s->octave = p->octave;
                        s->brother = r = new();
                        i = (findindex(currentscale,p->note)+6)%7;
                        r->note = scale_tab[currentscale][i].note;
                        r->accidental = scale_tab[currentscale][i].accidental
                                                                + chrom;
                        if ( p->note == 'C' )
                                r->octave = p->octave -1;
                        else
                                r->octave = p->octave;
                        q->brother = new();
                        q->brother->note = SEQ;
                        q->brother->brother = p;
                        q->brother->child = s;
        }

}

/***************************
dim.c
***************************/

#include "extern.h"
#include <stdio.h>

dim(old,n)
TNODEPTR        old;
int     n;
{
TNODEPTR        p,
                q,
                new();
int     i;

if   ( old->note != SEQ ){
        error ( "dim", "tree is not in the proper form");
        return ( 0 );
}

if   ( n < 0 ){
        error ( "dim", "invalid argument");
        return ( 0 );
```

```
/* check that there are enough notes */

for ( i = 0,p = old->child; p!= NULL && i < n ; p = p->brother,i++ )
        if ( !(p->note >= 'A'  && p->note <= 'G') ){
                error ( "dim", "tree is not in the proper form");
                return ( 0 );
        }

if ( i != n ){
        error ( "dim", "tree is not in the proper form");
        return ( 0 );
}

/* build replacement tree */

p = old->child;

p->child = q = new();
copy ( p,q);
p->note = SEQ;
q->child = NULL;

for ( i = 0 ; i < n-1; i++ ) {
        p = p->brother;
        q = q->brother = new();
        copy ( p,q );
        q->brother = NULL;
        q->child = NULL;
}


}

/*****************************
equalize.c
*****************************/

#include "extern.h"
#include <stdio.h>

equalize(old)
TNODEPTR        old;
{
TNODEPTR        p,
                q,
                r;

if ( (old->note != SIM && old->note != SEQ) ||  !old->child) {
        error ("equalize", "tree is not in proper form" );
        return ( 0 );
}


for ( p = old->child; p ;)
        if ( p->note == old->note ){
                if ( p == old->child )
                        old->child = p->child;
                else
                        q->brother = p->child;
                for ( r = p->child; r->brother; r = r->brother );
                r->brother = p->brother;
                p->child = NULL;
                p->brother = NULL;
                tr_free( p );
```

```
                p = r->brother;
                q = r;
        } else {
                q = p;
                p = p->brother;
        }
}

/*************************************
extern.c
*************************************/

#include        "def.h"
#include        "extern.h"


char    buf[BUFSIZE];
char    *gargv[MAXARGS];
int     gargc;

TNODEPTR        root;
TNODEPTR        cur_tree;

int     currentscale = 0;
int     chrom = 0;

struct scales   scale_tab[][7] = {
'C',0, 'D',0, 'E',0, 'F',0, 'G',0, 'A',0, 'B',0,
'G',0, 'A',0, 'B',0, 'C',0, 'D',0, 'E',0, 'F',1,
'D',0, 'E',0, 'F',1, 'G',0, 'A',0, 'B',0, 'C',1,
'A',0, 'B',0, 'C',1, 'D',0, 'E',0, 'F',1, 'G',1,
'E',0, 'F',1, 'G',1, 'A',0, 'B',0, 'C',1, 'D',1,
'B',0, 'C',1, 'D',1, 'E',0, 'F',1, 'G',1, 'A',1,
'F',1, 'G',1, 'A',1, 'B',0, 'C',1, 'D',1, 'E',1
};

int     verbose = 0;            /* default value for verbose is false  */

char    *PS1;
TNODEPTR        focuslist[128];
int     focusindex = 0;

char    last[132];
char    prev[132];


/*************************************
fillin.c
*************************************/

#include        "extern.h"
#include        <stdio.h>

TNODEPTR        fillin( high , low)
TNODEPTR        high,
                low;
{
TNODEPTR        p,
                q,
                new();

int             index,
                octave;

if ( (index = findindex ( currentscale, high->note ) ) < 0  ) {
```

```
        return ( 0 );
}

octave = high->octave;
p = q = new();

while ( (scale_tab[currentscale][index].note != low->note ) ||
             ( octave != low->octave ) ){
        if (scale_tab[currentscale][index].note == 'C' )
                --octave;
        index = ind_dec(index);

        q->note = scale_tab[currentscale][index].note;
        q->accidental = scale_tab[currentscale][index].accidental + chrom;
        q->octave = octave;

        if (scale_tab[currentscale][index].note == 'C' ){
                if (scale_tab[currentscale][ind_dec(index)].note == low->note
                                                && octave - 1 == low->octave)
                        break;
        } else
                if (scale_tab[currentscale][ind_dec(index)].note == low->note
                                                && octave  == low->octave)
                        break;
        q->brother = new();
        q = q->brother;
}
return( p );
}



TNODEPTR        upfold ( low, high )
TNODEPTR        low,
                high;
{
TNODEPTR        p,
                q,
                new();

int             index,
                octave;

if ( (index = findindex ( currentscale, low->note ) ) < 0  )
        error ( "upfold", "index not in scale table " );
octave = low->octave;
p = q = new();

while ( (scale_tab[currentscale][index].note != high->note ) ||
             ( octave != high->octave ) ){
        if (scale_tab[currentscale][index].note == 'B' )
                ++octave;
        index = ++index % 7;

        q->note = scale_tab[currentscale][index].note;
        q->accidental = scale_tab[currentscale][index].accidental + chrom;
        q->octave = octave;

        if (scale_tab[currentscale][index].note == 'B' ){
                if (scale_tab[currentscale][(index+1)%7].note == high->note
                                                && octave + 1 == high->octave
                        break;
        } else
                if (scale_tab[currentscale][(index+1)%7].note == high->note
                                                && octave  == high->octave)
```

```
        q->brother = new();
        q = q->brother;
}
return( p );
}

ind_dec(index)
int     index;
{
if ( index == 0 )
        return ( 7 );
else
        return ( index - 1 );
}



/************ flat ()  ******************/

flat(p)
TNODEPTR        p;
{

if ( p->note >= 'A' && p->note <= 'G' )
        --(p->accidental);
else
        error ("flat", "not in proper form" );

}


/*************************************
focus.c
*************************************/

#include        "extern.h"
#include        <stdio.h>

focus( offset )
int     offset;
{
if ( cur_tree->note == SIM || cur_tree->note == SEQ )
        if ( --offset > 0 && cur_tree->child )
                cur_tree = cur_tree->child;
while ( --offset > 0 && cur_tree->brother )
        cur_tree = cur_tree->brother;

}



#include        <stdio.h>
#include        "extern.h"

extend ( i )
int     i;
{
TNODEPTR        p,
                q,
                new();
int     n;

if ( !( cur_tree->note >= 'A' && cur_tree->note <= 'G' ) ) {
        error ( "extend", "can not extend this note " );
        return ( 0 );
```

```
if (  i < 2 ) {
        execerror ( "extend", "bad parameter" ,gargv[1] );
        return ( 0 );
}

p = new();
copy ( cur_tree, p );
p->child = NULL;
p->brother = NULL;
i--;

cur_tree->note = SEQ;
cur_tree->child = p;

for ( n = 0; n < i; n++){
        q = new();
        copy ( p,q );
        p = p->brother = q;
}

return ( i );

}

/****************************************
free.c
****************************************/


#include <stdio.h>
#include "extern.h"


tr_free( p )
TNODEPTR        p;
{

if ( p->child)
        tr_free(p->child);
if ( p->brother)
        tr_free(p->brother);
free ( p );

}


/****************************************
func.c
****************************************/

#include        <stdio.h>
#include        "def.h"


static struct functab {
        char    *name;
        char    *abbreviation;
        char    *synopsis;
} functab[] = {
        "test", "t", "",
        "nature", "n", "",
        "scale", "s",  "",
        "transpose", "tr",  "",
        "ursatz", "u", "",
        "focus", "f",  ""
```

```
        "reset", "r", "",
        "passingtone", "pt", "",
        "binary", "b", "",
        "clear", "c", "",
        "help", "h", "",
        "ternary", "te", "",
        "extend", "e", "",
        "auskomp", "a", "",
        "rauskomp","ra", "",
        "ua", "ua", "",
        "la", "l", "",
        "da", "d", "",
        "sharp", "sh", "",
        "flat", "fl", "",
        "parallel", "p", "",
        "remove", "re", "",
        "equalize", "eq", "",
        "dim","di", "",
        "write","w", "",
        "read","rd", "",
        "macro","m", "",
        "undo","un", "Undo last function",
        0 , 0 , 0
};

/********  help  ***********/
help()
{
int     c,
        i;

        printf( "FUNCION NAME          ABBREVIATION\tSYNOPSIS\n");
        for (i = 0; functab[i].name ; i++ ){
                printf( "     %s ",functab[i].name );
                if (strlen(functab[i].name) > 9)
                        printf("\t");
                else
                        printf("\t\t");
                printf( " %s",functab[i].abbreviation);
                printf( "\t%s",functab[i].synopsis);
                if ( !(i % 22) && ( i != 0 ) ){
                        printf ( "\nStrike <Enter> to continue...");
                        c = getchar();
                } else
                        printf ( "\n" );
        }
}



/******** getfunction ******/

/* look in functab for s, return index */

getfunction( s )
char    *s;

{
        int i;
        for (i = 0; functab[i].name &&  (strcmp( s, functab[i].name )!=0);i++);
        if ( functab[i].name )
                return ( i );
        else {
                for (i = 0; functab[i].abbreviation &&
                                        strcmp( s, functab[i].abbreviation)!=0;i++);
```

```
                        return ( i );
        }

        return ( i );
}


/******** error ******/

error ( function, message )             /* 2 parameters - magic function number
                                                and error message
*/
char    *function,
        *message;
{
        execerror ( function, message,  (char *) 0 );
}



/******** execerror ******/

execerror ( function, message, instance )       /* 3 parameters - last one is
                                                        example */
char    *function,
        *message,
        *instance;
{
        printf( "%s: %s", function, message );
        if (instance)
                printf( " %s", instance );

        printf ( "\n" );
}



/*****************************
gettoken.c
*****************************/

#include <stdio.h>

gettoken ( rbuf, infile )
char    *rbuf;
FILE    *infile;
{
int     c,
        i;

while ( (( c=getc(infile) ) != EOF ) && ( c == '\n' || c == '\b' || c == '\t'
                                                || c == ' ') );

if ( c != EOF ) {
        rbuf[0] = c;
        for ( i = 1; ( ( c=getc(infile) ) != EOF ) && !( c == '\n' || c == '\b'
                        || c == '\t' || c == ' ' || c == ')'  || c == '(');i++)
                rbuf[i] = c;
        rbuf[i] = '\0';
        ungetc(c,infile);               /* push c back on infile -delimiter */
                                        /* could be '(' or ')'              */
} else {
        rbuf[0] = '\0';
}
```

```
/*************************************
getword.c
*************************************/

#include        <stdio.h>
#include        "def.h"

extern  char    *gargv[MAXARGS];
extern  int     gargc;

char    *getword (s)
        char    *s;
{

        gargv[gargc] = s;
        if ( *s != '\n' ){
                gargv[gargc] = s;
                gargc++;
        };
        while ( *s != ' ' &&  *s != '\t' &&  *s != '\n' )
                s++;
        return ( s );
}


/*************************************
init.c
*************************************/

#include        <stdio.h>
#include        "extern.h"


init ()
{
root = NULL;
cur_tree = NULL;
focusindex  = 0;
focuslist[0] = NULL;
PS1 = "prog ==>  ";
strcpy ( last, "H1" );
strcpy ( prev, "H2" );
}



/*************************************
key.c
*************************************/


#include        <stdio.h>
#include        "extern.h"

static char key[] = { 'C', 'G', 'D', 'A', 'E', 'B', 'F', '\0'  };
char keys[] = { 'C', 'D', 'E', 'F', 'G', 'A', 'B', '\0'  };

findkey ( tonic )
char    tonic;
{
int     i;

for ( i = 0; key[i] ; i++ )
```

```
        if ( key[i] == tonic )
                return (i);
return ( -1 );
}



getinterval( low, high )
char    low,
        high;
{

int     h,
        l,
        i;

l = 0;
h = 0;
i = 0;
while ( keys[l] != low && l < 7 ) l++;
while ( keys[h] != high && h < 7 ) h++;
if ( h >= 7 || l >= 7 )
        return ( -1 );

while ( l++ % 7 != h ) i++;
return ( ++i );
}


findindex ( scale, note )
int     scale;
char    note;
{

int     i;

for ( i = 0; i <= 6 ; i++ ){
        if ( scale_tab[scale][i].note == note )
                return ( i );
}
return ( -1 );
}


sibcount( p )
TNODEPTR        p;
{
int     i;

for ( i = 0; p->brother ; i++ )
        p = p->brother;
return ( ++i );
}


                /* if A is higher than B return > 0
                if A is equal to B return 0
                if A is lower than B return < 0 */
dist( first, second )
TNODEPTR        first,
                second;
{
int     a,
        b;
```

```
b = 0;

while ( keys[a] != first->note && a < 7 ) a++;
while ( keys[b] != second->note && b < 7 ) b++;
if ( b >= 7 || a >= 7 )
        return ( NULL );

return ( ((first->octave * 7 )+ a ) - ( ( second->octave * 7 )+ b ) );
}



/***************************
la.c
***************************/

#include <stdio.h>
#include "extern.h"

la (old )
TNODEPTR        old;
{
TNODEPTR        p,
                q,
                r,
                new();
int     i;

if ( old->note != SEQ || old->child == NULL ){
        error ( "lower auxiliary", "this tree is not in the proper form" );
        return ( 0 );
}

for ( q = old->child, p = q->brother; p != NULL; q = p, p = p->brother )
        if ( p->note == q->note && p->octave == q->octave &&
                                                p->note >= 'A'
                                                &&  p->note <= 'G' ){
                        r = new();
                        i = (findindex(currentscale,p->note)+6)%7;
                        r->note = scale_tab[currentscale][i].note;
                        r->accidental = scale_tab[currentscale][i].accidental
                                                                + chrom;
                        if ( p->note == 'C' )
                                r->octave = p->octave -1;
                        else
                                r->octave = p->octave;
                        q->brother = new();
                        q->brother->note = SEQ;
                        q->brother->brother = p;
                        q->brother->child = r;
        }

}

/***************************************
main.c
***************************************/

#include        <stdio.h>
#include        <ctype.h>
#include        "extern.h"

main()

{
char *cp;
```

```
init();
fprintf( stdout, PS1 );
while ( fgets( buf, sizeof(buf ), stdin ) ) {
        for ( cp = buf; *cp; cp++ )
                if ( isupper ( *cp ) ) *cp = tolower ( *cp );
        split( buf );
        if ( gargc > 0 )
                parse( gargc, gargv );
        fprintf( stdout, PS1 );
};
if ( root)
        tr_free( root );
if ( cur_tree )
        tr_free( cur_tree );
}

/**********************************
macro.c
**********************************/

#include "extern.h"
#include <stdio.h>
#include <ctype.h>
#include <process.h>

macro(s)
char    *s;
{
char    name[128],
        *cp;

int     i;

FILE    *infile,
        *fopen();

strcpy(name,s);

if ( (infile = fopen("macfile","r") ) == NULL ){
        execerror ("macro", "cannot open macro file","macfile");
        return ( 0 );
}

/* Look for ".de <macroname>" in macrofile.  If found, process macro */

while ( fgets ( buf, sizeof(buf),infile) ){
        for ( cp = buf; *cp; cp++ )
                if ( isupper ( *cp ) ) *cp = tolower ( *cp );
        split( buf );
        if ( !(strcmp(gargv[0],".de")) && !(strcmp(gargv[1],name)) ){
                while ( strcmp(gargv[0],".."){
                        fgets ( buf, sizeof(buf),infile );
                        for ( cp = buf; *cp; cp++ )
                                if ( isupper ( *cp ) ) *cp = tolower ( *cp );
                        split( buf );
                        if ( gargc > 0 && strcmp(gargv[0],"..") ){
                                fprintf(stdout,PS1);
                                for ( i = 0; i < gargc; i++ )
                                        fprintf(stdout,"%s ",gargv[i] );
                                fprintf( stdout, "\n" );
                                parse( gargc, gargv );
                        }
                }
                fclose ( infile );
                return(1);
```

```
}

execerror("macro","could not find macro",name);
fclose (infile);
return(0);
}



/**********************************
macwrite.c
**********************************/

macwrite( macname )
char    *macname;
{
char    name[128],
        *cp;

int     i;

FILE    *infile,
        *outfile,
        *fopen();

if ( (infile = fopen("macfile","r") ) == NULL ){
        execerror ("macro", "cannot open macro file","macfile");
        return ( 0 );
}

if ( (outfile = fopen("macfile2","w") ) == NULL ){
        execerror ("macro", "cannot open work file","macfile2");
        fclose (infile);
        return ( 0 );
}

strcpy(name,macname);

/* copy all lines in macrofile except the lines
        ".de <macroname>"
   through
        ".."
  to macfile2 */

while ( fgets ( buf, sizeof(buf),infile) ){
        for ( cp = buf; *cp; cp++ )
                if ( isupper ( *cp ) ) *cp = tolower ( *cp );
        split( buf );
        if ( !(strcmp(gargv[0],".de")) && !(strcmp(gargv[1],name)) )
                while ( strcmp(gargv[0],".."){
                        fgets ( buf, sizeof(buf),infile );
                        for ( cp = buf; *cp; cp++ )
                                if ( isupper ( *cp ) ) *cp = tolower ( *cp );
                        split( buf );
                }
        else{
                for ( i = 0, cp = buf; i < (gargc-1); cp++ )
                        if (*cp == NULL ){
                                *cp = ' ';
                                i++;
                        }
                strcat ( buf, "\n" );
                fputs( buf, outfile );
        }
}
```

```
/* got rid of old macro (if it already existed )
                Now add new macro */
strcpy( buf, ".de ");
strcat( buf, name);
strcat( buf, "\n");
fputs( buf, outfile);
fprintf( stdout, PS1 );
while ( fgets( buf, sizeof(buf ), stdin) &&
                                        !(buf[0]=='.' && buf[1]=='.') ) {
        for ( cp = buf; *cp; cp++ )
                if ( isupper ( *cp ) ) *cp = tolower ( *cp );
        split( buf );
        if ( gargc > 0 ){
                parse( gargc, gargv );
                for ( i = 0, cp = buf; i < (gargc-1); cp++ )
                        if (*cp == NULL ){
                                *cp = ' ';
                                i++;
                        }
                strcat ( buf, "\n" );
                fputs( buf, outfile );
        }
        fprintf( stdout, PS1 );
}
fputs ("..\n", outfile);        /* put on terminating ".." */

fclose (infile);
fclose (outfile);

/*  Outfile is still Macrofile2 - must transfer data to macrofile */

infile = fopen("macfile","w");
outfile = fopen("macfile2","r");

while ( fgets ( buf, sizeof(buf), outfile) )
        fputs(buf,infile);

fclose (infile);
fclose (outfile);


}/*  save()
   This function will reset "last" and "prev" variables to appropriate
   file names and process appropriately.  - Last contains a snapshot of
   the tree after the most recently processing. Prev contains a snapshot
   of the tree before the processing captured in Last.
*/

#include "extern.h"
#include <stdio.h>

save()
{
FILE    *outfile,
        *fopen();

char    temp[132];

strcpy( temp, prev );
strcpy( prev, last );
strcpy( last, temp );

if ( !(outfile = fopen ( last, "w" ) ) ){
        execerror ( "write", "cannot open file", last );
        return ( 0 );
```

```
tree_write ( outfile, root );
fclose ( outfile );

}


undo()
{
char    temp[132];
TNODEPTR        rread();

strcpy( temp, prev );
strcpy( prev, last );
strcpy( last, temp );

tr_free ( root );

root = cur_tree = rread( last );
focusindex  = 0;
focuslist[focusindex++] = cur_tree;

}

/*************************************
nature.c
*************************************/

#include        <stdio.h>
#include        "extern.h"

TNODEPTR        nature ( i )
int i;
{
TNODEPTR        q,
                new();
int             toctave = 0;
int             octave;

if ( !(cur_tree->note == 'Z' || cur_tree->note == TONIC.note ) ){
        error( "nature" , "Can only transform the TONIC the nature form" );
        return(0);
}

cur_tree->note = SIM;
cur_tree->child = ( q = new() );

q->note = TONIC.note;
q->accidental = TONIC.accidental + chrom;
q->octave = toctave - 2;
q->brother = new();
q = q->brother;

q->note = TONIC.note;
q->accidental = TONIC.accidental + chrom;
q->octave = octave = toctave - 1;
q->brother = new();
q = q->brother;

q->note = DOMINANT.note;
q->accidental = DOMINANT.accidental + chrom;
q->octave = (octave += inc( TONIC.note, DOMINANT.note ));
q->brother = new();
q = q->brother;

q->note = TONIC.note;
```

```
q->accidental = TONIC.accidental + chrom;
q->octave = (octave += inc( DOMINANT.note , TONIC.note  ));
q->brother = new();
q = q->brother;

q->note = MEDIANT.note;
q->accidental = MEDIANT.accidental + chrom;
q->octave = (octave += inc( TONIC.note , MEDIANT.note  ));

if ( i == 5 || i == 8 ){
        q->brother = new();
        q = q->brother;
        q->note = DOMINANT.note;
        q->accidental = DOMINANT.accidental + chrom;
        q->octave = (octave += inc( MEDIANT.note , DOMINANT.note  ));
        if ( i == 8 ){
                q->brother = new();
                q = q->brother;
                q->note = TONIC.note;
                q->accidental = TONIC.accidental + chrom;
                q->octave = (octave += inc( DOMINANT.note , TONIC.note  ));
        };
};
return(cur_tree);
}


/************ num() ******************/

num( p )
TNODEPTR        p;

{

int     a;
extern char keys[];

a = 0;
while ( (keys[a] != p->note ) && a < 7 )
        a++;

if ( a >= 7 )
        return ( 0 );
else
        return ((p->octave * 7 )+ a );
}


/*************************************
new.c
*************************************/

#include        <stdio.h>
#include        "extern.h"

char    *emalloc(i)
unsigned        i;
{
char    *p,
        *malloc();

if ( p = malloc ( i ) )
        return ( p );
else{
        error ( gargv[0], "out of memory" );
```

```
}

}       /* end of emalloc       */


TNODEPTR  new()
{
TNODEPTR        p;

if ( p = (TNODEPTR) emalloc ( sizeof (TNODE) ) ){
        p->brother = NULL;
        p->child = NULL;
        p->accidental = 0 ;
        p->octave = 0 ;
        p->note = 'Z';
        return ( p );
}
else
        return ( (TNODEPTR) 0 );


}


/*************************************
oct_tran.c
*************************************/

#include        <stdio.h>
#include        "extern.h"

Otranspose( p, n )
TNODEPTR        p;
int             n;
{
int     i;

if ( p == NULL )
        return ( -1 );
i = 0;
FOREVER{
        p->octave += n;
        if ( p->brother ){
                p = p->brother;
                i++;
        }
        else
                return( i );
}
}



note_transpose( n )
int             n;
{

if ( cur_tree == NULL )
        return ( -1 );
cur_tree->octave += n;
        return( n );
}
```

```
pstone.c
************************************/


#include        <stdio.h>
#include        "extern.h"

pstone( old )
TNODEPTR        old;
{
TNODEPTR        p,
                q,
                n,
                new(),
                fillin(),
                upfold();
int             i;

/* check old tree to insure it is a sequence of notes   */

if ( old->note != SEQ ){
        error ( gargv[0] , "this chord is not in the proper form ");
        return( 0 );
};

q = old->child;

while ( q->brother )
        if( ( i = dist ( q, q->brother )) > 1 || i < -1 ){
                p = q->brother;
                q->brother = n = new();
                n->note = SEQ;
                n->brother = p;
                if ( i > 1)
                        n->child = fillin( q , p );
                else if ( i < -1 )
                        n->child = upfold( q , p );

                q = p;
        } else
                q = q->brother;
}


/**********************************
rauskomp.c
**********************************/

#include <stdio.h>
#include "extern.h"

rauskomp(old)
TNODEPTR        old;

{

int     a,
        gap,
        i,
        j,
        n,
        tempkeynumber,
        num_items;

TNODEPTR        n
```

```
                q,
                tempptr;

struct  {
        TNODEPTR        ptr;
        int       keynumber;
                        } sorttab[SORTSIZE];


/*   check that all nodes in subtree are actually notes */

if ( old->note != SIM ){
        error ( "rauskomp" , "must 'rauskomp' a simultaneity of notes");
        return (0 );
}
for ( p = old->child; p != NULL; p = p->brother )
        if ( !(p->note >= 'A' && p->note <= 'G' ) ){
                error ( "rauskomp" ,
                                        "must 'rauskomp' a simultaneity of notes");
                return (0 );
        }

/* checks out okay so proceed */

for ( p = old->child,i = 0; p != NULL; p = p->brother, i++ ){
        sorttab[i].ptr = p;
        sorttab[i].keynumber = num(p );
}

num_items = i;


for ( gap = num_items/2; gap > 0; gap /= 2 )
        for ( j = gap; j < num_items ; j++ )
                for ( i = j - gap; i >= 0; i -= gap ){
                        if ( sorttab[i].keynumber > sorttab[i+gap].keynumber )
                                break;
                        tempptr = sorttab[i].ptr;           /* swap values */
                        tempkeynumber = sorttab[i].keynumber;
                        sorttab[i].keynumber = sorttab[i+gap].keynumber;
                        sorttab[i].ptr = sorttab[i+gap].ptr;
                        sorttab[i+gap].ptr = tempptr;
                        sorttab[i+gap].keynumber = tempkeynumber;
                }



for ( i = 0; i < (num_items-1); i++ ){
        p = sorttab[i].ptr;
        p->brother = sorttab[i+1].ptr;
}

sorttab[num_items-1].ptr->brother = NULL;

old->note = SEQ;
old->child = sorttab[0].ptr;

}

/*********************************
read.c
*********************************/

#include <stdio.h>
#include "extern.h"
```

```
char    rbuf[128];

TNODEPTR        gettree(infile)
FILE    *infile;
{
TNODEPTR        new(),
                p,
                q;


gettoken(rbuf,infile);


if ( !(strcmp(rbuf, "(") ) )
        gettoken(rbuf,infile);          /* throw away first '(' in tree */

if ( !(strcmp(rbuf, "sim")) || !(strcmp(rbuf, "seq")) ){
        p = new();
        if ( !(strcmp(rbuf, "seq") ))
                p->note = SEQ;
        else if ( !(strcmp(rbuf, "sim") ) )
                p->note = SIM;
        q = p->child = gettree(infile);
        gettoken(rbuf,infile);
        if ( !(strcmp(rbuf, ")" ) ) ){   /* done! */
                return(p);
        } else {
                for ( ; strcmp(rbuf,")" ) != 0; ){
                        q = q->brother = gettree(infile);
                        gettoken(rbuf,infile);
                }
        /*ungetc(rbuf[0],infile);*/  /*push last ')' back on infile */
        }
} else  {                               /*  just a note */
        p = new();
        p->accidental = atoi(rbuf);
        gettoken(rbuf,infile);
        p->note = rbuf[0];
        gettoken(rbuf,infile);
        p->octave = atoi(rbuf);
        gettoken(rbuf,infile);                  /*  ")"  */
}

return(p);
}



TNODEPTR        rread(s)
char    *s;
{
FILE    *infile,
        *fopen();
TNODEPTR        gettree();

if ( !(infile = fopen ( s, "r" )) ){
        execerror ("read","cannot open input file", s);
        return ( (TNODEPTR) NULL );
}else
        return ( gettree(infile ) );

fclose(infile);

}
```

```
remove.c
*******************************/
#include "extern.h"
#include <stdio.h>

rremove (old)
TNODEPTR old;
{
TNODEPTR        findnode(),
                p;

p = findnode(root,old);

if ( p->child == old){
        p->child = old->brother;
        old->brother = NULL;
        tr_free (old);
}else if ( p->brother == old){
        p->brother = old->brother;
        old->brother = NULL;
        tr_free (old);
}else
        error( "remove", "illegal address" );

}


TNODEPTR findnode(p,target)
TNODEPTR        p,
                target;
{
TNODEPTR        q;

if ( p->child == target || p->brother == target )
        return ( p );
if ( p->child ){
        q = findnode ( p->child, target );
        if ( q && (q->child == target || q->brother == target) )
                return ( q );
}
if ( p->brother ){
        q = findnode ( p->brother, target );
        if ( q && (q->child == target || q->brother == target) )
                return ( q );
}
return ( NULL );
}

/*************************************
scale.c
*************************************/

#include        "extern.h"



scale ( i )
int i;
{
int     temp,
        dist;

                                        /*calculate distance from C natural */
currentscale = (dist = chrom * 7 + currentscale + i ) >= 0  ?
                                                dist % 7 : ( dist % 7 ) + 7 ;
```

```
/********** sharp ()  ****************/


sharp(p)
TNODEPTR        p;
{

if ( p->note >= 'A' && p->note <= 'G' )
        ++(p->accidental);
else
        error ("sharp", "not in proper form" );

}

/*************************************
split.c
*************************************/


#include        "def.h"
#include        <stdio.h>

        /*      return: > 1 - valid input
                        < 1 - blank line
                        ( -1 - error
        */


split ( buf )
char *buf;
{
        char    *s,
                *getword();
        extern int gargc;

        s = buf;
        gargc = 0;

        FOREVER {
                while ( *s == ' ' || *s == '\t' )
                        s++;
                s = getword(s);
                if (*s != '\n')
                        *s++ = '\0';
                else if (*s == '\n') {
                        *s = '\0';
                        return ( s - buf );
                }
        }
}


/*************************************
ternary.c
*************************************/


/*  The function's argument is a TNODEPTR.  The function transform a chord
     in the binary form into the corresponding ternary form.  */


#include        <stdio.h>
#include        "extern.h"
```

```
TNODEPTR        ternary( old )
TNODEPTR        old;
{
TNODEPTR        p,
                q,
                r,
                s,
                temp,
                new();

if ( ! ( isbinary ( old ) ) )
        return ( 0 );

        /*   position p so that tonic can be added as p's brother */

p = old->child->child->child;
while ( p->brother )
        p = p->brother;

q = new();

p->brother = q;

q->note = TONIC.note;
q->octave = p->octave + dec (SUPERTONIC.note, TONIC.note);
q->accidental = TONIC.accidental + chrom;


        /*   position r so that dominant can be added as r's brother */

r = old->child->child->brother->child;
while ( r->brother )
        r = r->brother;

q = new();

r->brother = q;

q->note = TONIC.note;
q->octave = r->octave + dec(DOMINANT.note, TONIC.note);
q->accidental = TONIC.accidental + chrom;

/*      define third component of ternary form and insert in proper place */

q = new();
q->note = SIM;

q->brother = old->child->brother;
old->child->brother = q;

q->child = new();
q->child->brother = new();

copy (p,q->child);
copy (r,q->child->brother);

return (old);
}


/***************************
ua.c
***************************/
```

```
#include "extern.h"

ua (old )
TNODEPTR        old;
{
TNODEPTR        p,
                q,
                r,
                new();
int     i;

if ( old->note != SEQ || old->child == NULL ){
        error ( "upper auxiliary", "this tree is not in the proper form" );
        return ( 0 );
}

for ( q = old->child, p = q->brother; p != NULL; q = p, p = p->brother )
        if ( p->note == q->note && p->octave == q->octave &&
                                                p->note >= 'A'
                                                &&  p->note <= 'G' ){
                        r = new();
                        i = (findindex(currentscale,p->note)+1)%7;
                        r->note = scale_tab[currentscale][i].note;
                        r->accidental = scale_tab[currentscale][i].accidental
                                                                + chrom;
                        if ( p->note == 'B' )
                                r->octave = p->octave +1;
                        else
                                r->octave = p->octave;
                        q->brother = new();
                        q->brother->note = SEQ;
                        q->brother->brother = p;
                        q->brother->child = r;
        }

}

/*************************************
unfold.c
*************************************/

#include        "extern.h"
#include        <stdio.h>

TNODEPTR        unfold ( high, low )
TNODEPTR        high,
                low;
{
TNODEPTR        p,
                q,
                new();

int             index,
                octave;

if ( (index = findindex ( currentscale, high->note ) ) < 0  )
        error ( "unfold", "index not in scale table " );
octave = high ->octave;
p = q = new();
copy ( high, q );
if ( q->note == 'C' )
        --octave;

while ( ( q->note != low->note ) || ( q->octave != low->octave ) ){
        if ( index == 0 )
                index += 6;
```

```
                --index;
        q->brother = new();
        q = q->brother;
        q->note = scale_tab[currentscale][index].note;
        q->accidental = scale_tab[currentscale][index].accidental + chrom;
        q->octave = octave;
        if ( q->note == 'C' )
                --octave;
}
return( p );
}



/*************************************
ursatz.c
*************************************/


#include        <stdio.h>
#include        "extern.h"

ursatz( old )
TNODEPTR        old;
{
TNODEPTR        p,
                q,
                treble,
                bass,
                brother(),
                new(),
                unfold();
int             octave,
                n;
char            note;

/* check old tree to insure it is in the form of a chord of nature */
if ( ! (n = isnature(old)) ){
        error ( "ursatz", "this chord is not in proper form" );
        return ( 0 );
}
/* this is in the proper form so unfold into ursatz form */


treble = new();
bass = new();

treble->note = SEQ;
bass->note = SEQ;

treble->brother = bass;

treble->child  = unfold (brother(n-1 , old->child), brother(3, old->child));

bass->child = q = new();
copy (brother(1, old->child), q );
q->brother = new();

q = q->brother;
copy (brother(2, old->child), q );
q->brother = new();

q = q->brother;
copy (brother(1, old->child), q );
```

```
tr_free (cur_tree->child);
cur_tree->child = treble;
}


/*************************************
write.c
*************************************/

#include <stdio.h>
#include "extern.h"

int     indent  = 0;

tr_write( p )
TNODEPTR        p;
{
        tree_write ( stdout, p );
}


toutput ( s )
char    *s;
{
FILE    *outfile,
        *fopen();

        if ( !(outfile = fopen ( s, "w" ) ) ){
                execerror ( "write", "cannot open file", s );
                return ( 0 );
        }

        tree_write ( outfile, cur_tree );
        fclose ( outfile );
}
```

```
/*************************************
isbinary.c
*************************************/



#include        <stdio.h>
#include        "extern.h"

isbinary( old )
TNODEPTR        old;
{
TNODEPTR        p,
                q;
int             octave;

if ( !(q = old ->child) || !(q = q->brother ) || !(q = q->child) ||
                                        !(q   q->brother ) || !(q = q->child) )
        return( 0 );
                                                /* snake through to where */
                                                   /* the tonic should be */
if ( findkey( q->note ) != currentscale )
        return( 0 );


if (old->note != SEQ )
        return( 0 );

if (( p = old->child )->note != SIM )
        return( 0 );

if ( ( p = p->child )->note != SEQ )
        return( 0 );

if ( ( q = p->brother)->note != SEQ )
        return( 0 );

if ( !(p = p->child))
        return( 0 );
if ( p->note == TONIC.note){    /*check treble notes <<< starts on tonic>>>*/
        octave = p->octave;

        if ( !(p = p->brother))
                return( 0 );
        if ( ( p->note != SUBTONIC.note ) ||
                    ( p->octave != (octave += dec(TONIC.note, SUBTONIC.note))))
                return( 0 );

        if ( !(p = p->brother))
                return( 0 );
        if ( ( p->note != SUPERDOMINANT.note ) || ( p->octave != (octave += dec(SUBTONIC.note, SUPERDOMINANT.note))))
                return( 0 );

        if ( !(p = p->brother))
                return( 0 );
        if ( ( p->note != DOMINANT.note ) || ( p->octave != (octave += dec(SUPERDOMINANT.note, DOMINANT.note))))
                return( 0 );

        if ( !(p = p->brother))
                return( 0 );
        if ( ( p->note != SUBDOMINANT.note ) || ( p->octave != (octave += dec(DOMINANT.note, SUBDOMINANT.note))))
                return( 0 );

        if ( !(p = p->brother))
                return( 0 );
```

```
                return ( 0 );

        if ( !(p = p->brother))
                return( 0 );
        if ( ( p->note != SUPERTONIC.note ) || ( p->octave != (octave += dec(MEDIANT.note, SUPERTONIC.note))))
                return ( 0 );


        if ( p->brother)
                return( 0 );

} else if ( p->note == DOMINANT.note ) {        /* <<< start on dominant down>>>*/
        octave = p->octave;

        if ( !(p = p->brother))
                return( 0 );
        if ( ( p->note != SUBDOMINANT.note ) || ( p->octave != (octave += dec(DOMINANT.note, SUBDOMINANT.note))))
                return( 0 );

        if ( !(p = p->brother))
                return( 0 );
        if ( ( p->note != MEDIANT.note ) || ( p->octave != (octave += dec(SUBDOMINANT.note, MEDIANT.note))))
                return( 0 );

        if ( !(p = p->brother))
                return( 0 );
        if ( ( p->note != SUPERTONIC.note ) || ( p->octave != (octave += dec(MEDIANT.note, SUPERTONIC.note))))
                return( 0 );

        if ( p->brother)
                return( 0 );

} else if ( p->note == MEDIANT.note ) {         /*<<< start on mediant down >>>*/
        octave = p->octave;

        if ( !(p = p->brother))
                return( 0 );
        if ( ( p->note != SUPERTONIC.note ) || ( p->octave != (octave += dec(MEDIANT.note, SUPERTONIC.note))))
                return( 0 );



if ( p->brother)
                return( 0 );

} else
                return( 0 );

                                        /*check bass notes */

        if ( !(q = q->child))
                return( 0 );
if ( ( q->note != TONIC.note )|| ( q->octave != ( octave -= 1) ))
                return( 0 );
octave = q->octave;

        if ( !(q = q->brother))
                return( 0 );
if ( ( q->note != DOMINANT.note ) || ( q->octave != (octave +=inc(TONIC.note, DOMINANT.note))))
                return( 0 );

        if ( q->brother)
                return( 0 );

/* reset p and q and check other ursatz-like sub-tree */
```

```
if (( p = old->child->brother )->note != SIM )
                return( 0 );

if ( ( p = p->child )->note != SEQ )
                return( 0 );

if ( ( q = p->brother)->note != SEQ )
                return( 0 );

 f ( !(p - p->child))
                return( 0 );
 f ( p->note == TONIC.note){    /*check treble notes <<< starts on tonic>>>*/
        octave = p->octave;

        if ( !(p = p->brother))
                return( 0 );
        if ( ( p->note != SUBTONIC.note ) || ( p->octave != (octave += dec(TONIC.note, SUBTONIC.note))))
                return( 0 );

        if ( !(p = p->brother))
                return( 0 );
        if ( ( p->note != SUPERDOMINANT.note ) || ( p->octave != (octave += dec(SUBTONIC.note, SUPERDOMINANT.note))))
                return( 0 );

        if ( !(p = p->brother))
                return( 0 );
        if ( ( p->note != DOMINANT.note ) || ( p->octave != (octave += dec(SUPERDOMINANT.note, DOMINANT.note))))
                return( 0 );

        if ( !(p = p->brother))
                return( 0 );
        if ( ( p->note != SUBDOMINANT.note ) || ( p->octave != (octave += dec(DOMINANT.note, SUBDOMINANT.note))))
                return( 0 );

        if ( !(p = p->brother))
                return( 0 );
        if ( ( p->note != MEDIANT.note ) || ( p->octave != (octave += dec(SUBDOMINANT.note, MEDIANT.note))))
                return ( 0 );

        if ( !(p = p->brother))
                return( 0 );
        if ( ( p->note != SUPERTONIC.note ) || ( p->octave != (octave += dec(MEDIANT.note, SUPERTONIC.note))))
                return ( 0 );


        if ( !(p = p->brother))
                return( 0 );
        if ( ( p->note != TONIC.note ) || ( p->octave != (octave += dec(SUPERTONIC.note, TONIC.note))))
                return( 0 );

 } else if ( p->note == DOMINANT.note ) {        /* <<< start on dominant down>>>*/
        octave = p->octave;

        if ( !(p = p->brother))
                return( 0 );
        if ( ( p->note != SUBDOMINANT.note ) || ( p->octave != (octave += dec(DOMINANT.note, SUBDOMINANT.note))))
                return( 0 );

        if ( !(p = p->brother))
                return( 0 );
        if ( ( p->note != MEDIANT.note ) || ( p->octave != (octave += dec(SUBDDMINANT.note, MEDIANT.note))))
                return( 0 );

        if ( !(p = p->brother))
```

```
        if ( ( p->note != SUPERTONIC.note ) || ( p->octave != (octave += dec(MEDIANT.note, SUPERTONIC.note))))
                return( 0 );

        if ( !(p = p->brother))
                return( 0 );
        if ( ( p->note != TONIC.note ) || ( p->octave != (octave += dec(SUPERTONIC.note, TONIC.note))))
                return( 0 );

} else if ( p->note == MEDIANT.note ) {         /*<<< start on mediant down >>>*/
        octave = p->octave;

        if ( !(p = p->brother))
                return( 0 );
        if ( ( p->note != SUPERTONIC.note ) || ( p->octave != (octave += dec(MEDIANT.note, SUPERTONIC.note))))
                return( 0 );



if ( !(p = p->brother))
                return( 0 );
        if ( ( p->note != TONIC.note ) || ( p->octave != (octave += dec(SUPERTONIC.note, TONIC.note))))
                return( 0 );

} else
                return( 0 );

                                        /*check bass notes */

        if ( !(q = q->child))
                return( 0 );
if ( ( q->note != TONIC.note ) || ( q->octave != ( octave -= 1) ))
                return( 0 );
octave = q->octave;

        if ( !(q = q->brother))
                return( 0 );
if ( ( q->note != DOMINANT.note ) || ( q->octave != (octave +=inc(TONIC.note, DOMINANT.note))))
                return( 0 );

        if ( !(q = q->brother))
                return( 0 );
if ( ( q->note != TONIC.note ) || ( q->octave != (octave += dec(DOMINANT.note, TONIC.note))))
                return( 0 );


/*      success         */
return ( 1 );
}



                return( 0 );

/*      success         */
return ( 1 );
}

/************************************
isnature.c
************************************/

#include        <stdio.h>
#include        "extern.h"

isnature ( old )
TNODEPTR        old;
```

```
TNODEPTR        p,
                q;

int             octave,
                n;
char            note;

/* check old tree to insure it is in the form of a chord of nature */
if ( old->note != SIM || !(old->child) ){
        return (0);
)
q = old->child;
if ( findkey( q->note ) != currentscale ){
        return( 0 );
);

if (!( (n = sibcount( old->child )) == 5 || n == 6 || n == 7 )){
        return ( 0 );
)

octave = q->octave;

if ( q->note != TONIC.note  ){
        return ( 0 );
)

q = q->brother;
if ( q->note != TONIC.note || q->octave != ++octave  ){
        return ( 0 );
}


q = q->brother;
if ( q->note != DOMINANT.note || q->octave !=(octave += inc(TONIC.note ,DOMINANT.note )) ){
        return ( 0 );
)

q = q->brother;
if ( q->note != TONIC.note || q->octave !=(octave += inc(DOMINANT.note ,TONIC.note )) ){
        return ( 0 );
}


q = q->brother;
if ( q->note != MEDIANT.note || q->octave !=(octave += inc(TONIC.note ,MEDIANT.note )) ) {
        return ( 0 );
)

if ( n == 6 || n == 7 ){
        q = q->brother;
        if ( q->note != DOMINANT.note || q->octave != ( octave += inc(MEDIANT.note ,DOMINANT.note )) ){
                return ( 0 );
        }

        if ( n == 7 ){
                q = q->brother;
                if ( q->note != TONIC.note || q->octave != (octave += inc(DOMINANT.note ,TONIC.note )) ){
                        return ( 0 );
                }
        }
)

return ( n );       /*      SUCCESS !!      */

}
```

```
/*************************************
isternary.c
*************************************/



#include        <stdio.h>
#include        "extern.h"

isternary( old )
TNODEPTR        old;
{
TNODEPTR        p,
                q;
int             octave,
                startnote;

if ( !(q = old ->child) || !(q = q->child) || !(q = q->brother ) || !(q = q->child) )
        return( 0 );
                                                /* snake through to where */
                                                /* the tonic should be */
if ( findkey( q->note ) != currentscale )
        return( 0 );


                                                /*   check accidental   */
if ( q->accidental !=  TONIC.accidental + chrom )
        return( 0 );

if (old->note != SEQ )
        return( 0 );

if (( p = old->child )->note != SIM )
        return( 0 );

if ( ( p = p->child )->note != SEQ )
        return( 0 );

if ( ( q = p->brother)->note != SEQ )
        return( 0 );

if ( !(p = p->child))
        return( 0 );
if ( p->note == TONIC.note){    /*check treble notes <<< starts on tonic>>>*/
        startnote = 8;
        octave = p->octave;

        if ( !(p = p->brother))
        return( 0 );
        if ( ( p->note != SUBTONIC.note ) || ( p->octave != (octave += dec(TONIC.note, SUBTONIC.note))))
        return( 0 );

        if ( !(p = p->brother))
        return( 0 );
        if ( ( p->note != SUPERDOMINANT.note ) || ( p->octave != (octave += dec(SUBTONIC.note, SUPERDOMINANT.note))))
        return( 0 );

        if ( !(p = p->brother))
        return( 0 );
        if ( ( p->note != DOMINANT.note ) || ( p->octave != (octave += dec(SUPERDOMINANT.note, DOMINANT.note))))
        return( 0 );

        if ( !(p = p->brother))
        return( 0 );
        if ( ( p->note != SUBDOMINANT.note ) || ( p->octave != (octave += dec(DOMINANT.note, SUBDOMINANT.note))))
        return( 0 );
```

```
        if ( !(p = p->brother))
        return( 0 );
        if ( ( p->note != MEDIANT.note ) || ( p->octave != (octave += dec(SUBDOMINANT.note, MEDIANT.note))))
                return ( 0 );

        if ( !(p = p->brother))
        return( 0 );
        if ( ( p->note != SUPERTONIC.note ) || ( p->octave != (octave += dec(MEDIANT.note, SUPERTONIC.note))))
                return ( 0 );


        if ( !(p = p->brother))
        return( 0 );
        if ( ( p->note != TONIC.note ) || ( p->octave != (octave += dec(SUPERTONIC.note, TONIC.note))))
        return( 0 );

        if ( p->brother)
        return( 0 );

} else if ( p->note == DOMINANT.note ) {        /* <<< start on dominant down>>>*/
        startnote = 5;
        octave = p->octave;

        if ( !(p = p->brother))
        return( 0 );
        if ( ( p->note != SUBDOMINANT.note ) || ( p->octave != (octave += dec(DOMINANT.note, SUBDOMINANT.note))))
        return( 0 );

        if ( !(p = p->brother))
        return( 0 );
        if ( ( p->note != MEDIANT.note ) || ( p->octave != (octave += dec(SUBDOMINANT.note, MEDIANT.note))))
        return( 0 );

        if ( !(p = p->brother))
        return( 0 );
        if ( ( p->note != SUPERTONIC.note ) || ( p->octave != (octave += dec(MEDIANT.note, SUPERTONIC.note))))
        return( 0 );

        if ( !(p = p->brother))
        return( 0 );
        if ( ( p->note != TONIC.note ) || ( p->octave != (octave += dec(SUPERTONIC.note, TONIC.note))))
        return( 0 );

        if ( p->brother)
        return( 0 );

} else if ( p->note == MEDIANT.note ) {         /*<<< start on mediant down >>>*/
        startnote = 3;
        octave - p->octave;

        if ( !(p = p->brother))
        return( 0 );
        if ( ( p->note != SUPERTONIC.note ) || ( p->octave != (octave += dec(MEDIANT.note, SUPERTONIC.note))))
        return( 0 );

        if ( !(p = p->brother))
        return( 0 );
        if ( ( p->note != TONIC.note ) || ( p->octave != (octave += dec(SUPERTONIC.note, TONIC.note))))
        return( 0 );



if ( p->brother)
        return( 0 );
```

```
        return( 0 );

                                        /*check bass notes */

        if ( !(q = q->child))
        return( 0 );
if ( ( q->note != TONIC.note )|| ( q->octave != ( octave -= 1) ))
        return( 0 );
octave = q->octave;

        if ( !(q = q->brother))
        return( 0 );
if ( ( q->note != DOMINANT.note ) || ( q->octave != (octave +=inc(TONIC.note, DOMINANT.note))))
        return( 0 );

        if ( !(q = q->brother))
        return( 0 );
if ( ( q->note != TONIC.note ) || ( q->octave != (octave += dec(DOMINANT.note, TONIC.note))))
        return( 0 );

        if ( q->brother)
        return( 0 );

/* reset p and q and check next sub-tree */

if (( p - old->child->brother )->note != SIM )
        return( 0 );

if ( !(p = p->child) || !( p->note == SUPERTONIC.note ))
        return( 0 );
if ( !(p = p->brother) || !( p->note == DOMINANT.note))
        return( 0 );


/* reset p and q and check other ursatz-like sub-tree */


if (( p = old->child->brother->brother )->note != SIM )
        return( 0 );

if ( ( p = p->child )->note != SEQ )
        return( 0 );

if ( ( q = p->brother)->note != SEQ )
        return( 0 );

if ( !(p = p->child))
        return( 0 );
if ( startnote == 8 ){    /*check treble notes <<< starts on tonic>>>*/
        if ( p->note != TONIC.note)
        return( 0 );

        octave = p->octave;

        if ( !(p = p->brother))
        return( 0 );
        if ( ( p->note != SUBTONIC.note ) || ( p->octave != (octave += dec(TONIC.note, SUBTONIC.note))))
        return( 0 );

        if ( !(p = p->brother))
        return( 0 );
        if ( ( p->note != SUPERDOMINANT.note ) || ( p->octave != (octave += dec(SUBTONIC.note, SUPERDOMINANT.note))))
        return( 0 );

        if ( !(p = p->brother))
```

```
        if ( ( p->note != DOMINANT.note ) || ( p->octave != (octave += dec(SUPERDOMINANT.note, DOMINANT.note))))
        return( 0 );

        if ( !(p = p->brother))
        return( 0 );
        if ( ( p->note != SUBDOMINANT.note ) || ( p->octave != (octave += dec(DOMINANT.note, SUBDOMINANT.note))))
        return( 0 );

        if ( !(p = p->brother))
        return( 0 );
        if ( ( p->note != MEDIANT.note ) || ( p->octave != (octave += dec(SUBDOMINANT.note, MEDIANT.note))))
                return ( 0 );

        if ( !(p = p->brother))
        return( 0 );
        if ( ( p->note != SUPERTONIC.note ) || ( p->octave != (octave += dec(MEDIANT.note, SUPERTONIC.note))))
                return ( 0 );


        if ( !(p = p->brother))
        return( 0 );
        if ( ( p->note != TONIC.note ) || ( p->octave != (octave += dec(SUPERTONIC.note, TONIC.note))))
        return( 0 );

} else if ( startnote == 5 ){        /* <<< start on dominant down>>>*/
        if ( p->note != DOMINANT.note )
        return( 0 );

        octave = p->octave;

        if ( !(p = p->brother))
        return( 0 );
        if ( ( p->note != SUBDOMINANT.note ) || ( p->octave != (octave += dec(DOMINANT.note, SUBDOMINANT.note))))
        return( 0 );

        if ( !(p = p->brother))
        return( 0 );
        if ( ( p->note != MEDIANT.note ) || ( p->octave != (octave += dec(SUBDOMINANT.note, MEDIANT.note))))
        return( 0 );

        if ( !(p = p->brother))
        return( 0 );
        if ( ( p->note != SUPERTONIC.note ) || ( p->octave != (octave += dec(MEDIANT.note, SUPERTONIC.note))))
        return( 0 );

        if ( !(p = p->brother))
        return( 0 );
        if ( ( p->note != TONIC.note ) || ( p->octave != (octave += dec(SUPERTONIC.note, TONIC.note))))
        return( 0 );

 else if ( startnote == 3 ){         /*<<< start on mediant down >>>*/
        if ( p->note != MEDIANT.note )
        return( 0 );

        octave = p->octave;

        if ( !(p - p->brother))
        return( 0 );
        if ( ( p->note != SUPERTONIC.note ) || ( p->octave != (octave += dec(MEDIANT.note, SUPERTONIC.note))))
        return( 0 );



f ( !(p = p->brother))
        return( 0 );
```

```
        return( 0 );

) else
        return( 0 );

                                        /*check bass notes */

        if ( !(q = q->child))
        return( 0 );
if ( ( q->note != TONIC.note ) || ( q->octave != ( octave -= 1) ))
        return( 0 );
octave = q->octave;

        if ( !(q = q->brother))
        return( 0 );
if ( ( q->note != DOMINANT.note ) || ( q->octave != (octave +=inc(TONIC.note, DOMINANT.note))))
        return( 0 );

        if ( !(q = q->brother))
        return( 0 );
if ( ( q->note != TONIC.note ) || ( q->octave != (octave += dec(DOMINANT.note, TONIC.note))))
        return( 0 );


/*      success         */
return ( 1 );
}


/************************************
isursatz.c
************************************/


#include        <stdio.h>
#include        "extern.h"

isursatz( old )
TNODEPTR        old;
{
TNODEPTR        p,
                q;
int             octave;

q = old->child->brother->child;

if ( findkey( q->note ) != currentscale )
        return( 0 );

if (old->note != SIM )
        return( 0 );

if ( ( p = old->child )->note != SEQ )
        return( 0 );

if ( ( q = p->brother)->note != SEQ )
        return( 0 );

if ( !(p = p->child))
        return( 0 );
if ( p->note == TONIC.note){    /*check treble notes */
        octave = p->octave;

        if ( !(p = p->brother))
                return( 0 );
```

```
            return( 0 );

        if ( !(p = p->brother))
            return( 0 );
        if ( ( p->note != SUPERDOMINANT.note ) || ( p->octave != (octave += dec(SUBTONIC.note, SUPERDOMINANT.note))))
            return( 0 );

        if ( !(p = p->brother))
            return( 0 );
        if ( ( p->note != DOMINANT.note ) || ( p->octave != (octave += dec(SUPERDOMINANT.note, DOMINANT.note))))
            return( 0 );

        if ( !(p = p->brother))
            return( 0 );
        if ( ( p->note != SUBDOMINANT.note ) || ( p->octave != (octave += dec(DOMINANT.note, SUBDOMINANT.note))))
            return( 0 );

        if ( !(p = p->brother))
            return( 0 );
        if ( ( p->note != MEDIANT.note ) || ( p->octave != (octave += dec(SUBDOMINANT.note, MEDIANT.note))))
            return ( 0 );

        if ( !(p = p->brother))
            return( 0 );
        if ( ( p->note != SUPERTONIC.note ) || ( p->octave != (octave += dec(MEDIANT.note, SUPERTONIC.note))))
            return ( 0 );


        if ( !(p = p->brother))
            return( 0 );
        if ( ( p->note != TONIC.note ) || ( p->octave != (octave += dec(SUPERTONIC.note, TONIC.note))))
            return( 0 );

        if ( p->brother)
            return( 0 );
    } else if ( p->note == DOMINANT.note ) {
        octave = p->octave;

        if ( !(p = p->brother))
            return( 0 );
        if ( ( p->note != SUBDOMINANT.note ) || ( p->octave != (octave += dec(DOMINANT.note, SUBDOMINANT.note))))
            return( 0 );

        if ( !(p = p->brother))
            return( 0 );
        if ( ( p->note != MEDIANT.note ) || ( p->octave != (octave += dec(SUBDOMINANT.note, MEDIANT.note))))
            return( 0 );

        if ( !(p = p->brother))
            return( 0 );
        if ( ( p->note != SUPERTONIC.note ) || ( p->octave != (octave += dec(MEDIANT.note, SUPERTONIC.note))))
            return( 0 );

        if ( !(p = p->brother))
            return( 0 );
        if ( ( p->note != TONIC.note ) || ( p->octave != (octave += dec(SUPERTONIC.note, TONIC.note))))
            return( 0 );
        if ( p->brother)
            return( 0 );

    } else if ( p->note == MEDIANT.note ) {
        octave = p->octave;

        if ( !(p = p->brother))
            return( 0 );
```

```
                return( 0 );

        if ( !(p = p->brother))
                return( 0 );
        if ( ( p->note != TONIC.note ) || ( p->octave != (octave += dec(SUPERTONIC.note, TONIC.note))))
                return( 0 );

        if ( p->brother)
                return( 0 );
} else
                                        /*check bass notes */

        if ( !(q = q->child))
        return( 0 );
if ( ( q->note != TONIC.note ) || ( q->octave != ( octave -= 1) ))
        return( 0 );
octave = q->octave;

        if ( !(q = q->brother))
        return( 0 );
if ( ( q->note != DOMINANT.note ) || ( q->octave != (octave +=inc(TONIC.note, DOMINANT.note))))
        return( 0 );

        if ( !(q = q->brother))
        return( 0 );
if ( ( q->note != TONIC.note ) || ( q->octave != (octave += dec(DOMINANT.note, TONIC.note))))
        return( 0 );
        if ( p->brother)

/**************************
parallel.c
**************************/

#include <stdio.h>
#include "extern.h"

parallel (old ,n)
TNODEPTR        old;
int     n;
{
TNODEPTR        p,
                q,
                r,
                s,
                new();
int     down,
        i;

if  ( n < 0 ){
        n = -n;
        down = 1;
}else
        down = 0;

n--;
if ( old->note != SEQ || old->child == NULL ){
        error ( "parallel", "this tree is not in the proper form" );
        return ( 0 );
}

for ( q = old->child, p = q->brother; q != NULL; ){
        if ( q->note >= 'A' &&  q->note <= 'G' ){
                        r = new();
                        if (down){
                                i = (findindex(currentscale,q->note)-(n%7))%7;
```

```
                        i += 7;
                }else
                        i = (findindex(currentscale,q->note)+n)%7;
                r->note = scale_tab[currentscale][i].note;
                r->accidental = scale_tab[currentscale][i].accidental
                                                        + chrom;
                if (down)
                        r->octave = q->octave - (n/7)
                                        + dec2(q->note,r->note);
                else
                        r->octave = q->octave + (n/7)
                                        + inc(q->note,r->note);
                s = new();
                copy (q,s);
                s->brother = r;
                q->note = SIM;
                q->octave = 0;
                q->child = s;
        }
        q = p;
        if ( p != NULL )
                p = p->brother;
}

}

/************************************
parse.c
************************************/

#include        <stdio.h>
#include        <ctype.h>
#include        "extern.h"



parse( gargc, gargv)
        int gargc;
        char *gargv[];
{
int     i;
NODEPTR         p,
                binary(),
                hang(),
                nature(),
                new(),
                rread();

switch ( getfunction( gargv[0])) {

        case AUSKOMP:
                auskomp(cur_tree);
                tr_write (cur_tree);
                save();
        break;

        case BINARY:
                if ( isursatz (cur_tree) ){
                        cur_tree = binary(cur_tree);
                        /*root = cur_tree;*/
                }
                else
                        error ( "binary", "this chord is not in the proper form");
                tr_write(cur_tree);
                save();
```

```
case CLEAR:
        cur_tree = NULL;
        focusindex = 0;
        focuslist[0] = NULL;
        if ( root )
                tr_free ( root );
        root = NULL;
        save();
break;

case DA:
        da ( cur_tree );
        tr_write ( cur_tree );
        save();
break;


case DIM:
        if (gargc != 2)
                error ("dim", "invalid argument" );
        else{
                dim ( cur_tree, atoi(gargv[1]) );
                tr_write ( cur_tree );
        }
        save();
break;


case EQUALIZE:
        if ( cur_tree );
                equalize ( cur_tree );
        tr_write ( cur_tree );
        save();
break;


case EXTEND:
        if ( gargc != 2 )
                error( "extend", "faulty syntax" );
        else
                extend( atoi(gargv[1]) );
        tr_write(cur_tree);
        save();
break;

case FLAT:
        flat ( cur_tree );
        tr_write ( cur_tree );
        save();
break;


case FOCUS:
        switch(gargc){

                case 1:
                        break;

        /* call focus to change focus upon reading each valid argument*/
                default:

'* check each string from gargv[n] to last
:ommand line argument to ensure that they
re all digits*/
```

```
                        for ( i=1; i< gargc ; i++ )
                                focus( atoi (gargv[i]) );
                        focusindex++;
                        focuslist[focusindex] = cur_tree;
                }
                break;
        }
        if (cur_tree)
                tr_write(cur_tree);
        save();
        break;

case HELP:
        help();
break;

case LA:
        la ( cur_tree );
        tr_write ( cur_tree );
        save();
break;


case MACRO:
        if (gargc == 2 )
                macro( gargv[1] );
        else if ( (gargc == 3) &&
                    ( !(strcmp(gargv[1], "write" ))  ||
                      !(strcmp(gargv[1], "WRITE" )) ) )
                macwrite (gargv[2]);
        else
                error ( "macro", "invalid format" );
        save();
break;


case NATURE:
        if ( gargc < 2 )
                error( "nature", "too few actual parameters" );
        else
                switch ( i = atoi(gargv[1]) ) {
                        case 3:
                        case 5:
                        case 8:    if ( root == NULL ){
                                        root = new();
                                        cur_tree = root;
                                        focuslist[0] = root;
                                        focusindex = 0;
                                        root->note = TONIC.note;
                                        root->octave = 0;
                                        nature( i );
                                } else if (cur_tree->note==TONIC.note
                                || cur_tree->note == SUPERTONIC.note
                            || cur_tree->note == MEDIANT.note
                            || cur_tree->note == SUBDOMINANT.note
                            || cur_tree->note == DOMINANT.note
                            || cur_tree->note == SUPERDOMINANT.note
                            || cur_tree->note == SUBTONIC.note ){
                                        nature ( i);
                                  } else
                                        execerror ("nature","current note is not diatonic in current scale");
                                        tr_write ( cur_tree );
                                        break;
                           default:     execerror( "nature", "invalid parameter", gargv[1]);
                                        break;
```

```
        save();
        break;

case PARALLEL:
        if ( gargc != 2 ){
                error ( "parallel", "not enough arguments" );
                tr_write ( cur_tree );
                break;
        }
        parallel(cur_tree, atoi(gargv[1]) );
        tr_write(cur_tree);
        save();
break;

case PASSTONE:
        pstone(cur_tree);
        tr_write(cur_tree);
        save();
break;

case RAUSKOMP:
        rauskomp(cur_tree);
        tr_write (cur_tree);
        save();
break;

case READ:
        if ( root == NULL ) {
                root = cur_tree = rread(gargv[1]);
                focusindex = 0;
                focuslist[focusindex++] = cur_tree;
        } else {
                p = cur_tree->brother;
                cur_tree->brother = rread(gargv[1]);
                cur_tree = cur_tree->brother;
                cur_tree->brother = p;
                focuslist[focusindex++] = cur_tree;
        }
        tr_write (cur_tree);
        save();
break;

case REMOV:
        if ( focusindex > 0 ){
                rremove(cur_tree);
                cur_tree = focuslist[--focusindex];
        } else if ( focusindex == 0 ){
                cur_tree = NULL;
                focuslist[0] = NULL;
                tr_free ( root );
                root = NULL;
        }
        tr_write (cur_tree);
        save();
break;

case RESET:
        if ( focusindex > 0 )
                cur_tree = focuslist[--focusindex];
        else if ( focusindex == 0 )
                cur_tree = focuslist[focusindex];
        tr_write(cur_tree);
break;

case SCALE:
```

```
            fprintf ( stdout, "current scale is: %c ", TONIC.note );
            fprintf ( stdout, "    accidental is: %d ", TONIC.accidental+chrom  );
            if  ( TONIC.accidental+chrom  > 0 )
                    for ( i = TONIC.accidental+chrom  ; i-- > 0; )
                            fprintf ( stdout, "%c", '#' );
            else if  ( TONIC.accidental+chrom    < 0 )
                    for ( i = TONIC.accidental+chrom  ; i++ < 0; )
                            fprintf ( stdout, "%c", 'b' );
            fprintf ( stdout, "\n" );
            break;
        }
        switch( *gargv[1] )(
            case 'c':
            case 'C':       currentscale - C;
                            chrom = 0;
                            break;
            case 'g':
            case 'G':       currentscale = G;
                            chrom = 0;
                            break;
            case 'd':
            case 'D':       currentscale = D;
                            chrom - 0;
                            break;
            case 'a':
            case 'A':       currentscale = A;
                            chrom = 0;
                            break;
            case 'e':
            case 'E':       currentscale = E;
                            chrom = 0;
                            break;
            case 'b':
            case 'B':       currentscale - B;
                            chrom = 0;
                            break;
            case 'f':
            case 'F':       currentscale = F;
                            chrom - -1;
                            break;
            case '-':       if ( gargc >2 )(
                                    scale (-(atoi(gargv[2])));
                                    break;
                            }
                            scale (atoi(gargv[1]));
                            break;
            case '+':       if ( gargc >2 )(
                                    scale (atoi(gargv[2]));
                                    break;
                             }
                             scale (atoi(++gargv[1]));
                             break;
            default:
                    execerror ( "scale", "illegal parameter" , ( gargc > 2 ? gargv[2] : gargv[1] ) );
                    break;
            }
            if ( verbose )
                    fprintf ( stdout, "current scale is %c", TONIC );
        save();
        break;

case SHARP:
        sharp ( cur_tree );
        tr_write ( cur_tree );
        save();
```

```
case TERNARY:
        ternary ( cur_tree );
        tr_write ( cur_tree );
        save();
break;

case TEST:
        test();
        break;

case TRANSPOSE:
        switch( gargc ){
                case 2:
                        note_transpose( atoi( gargv[1] ));
                        break;
                case 3:
                        if ( strcmp (gargv[1], "long") != 0 ){
                                execerror ( "transpose", "unknown argument", gargv[1] );
                                break;
                        }

                        Otranspose( cur_tree->child , atoi( gargv[2] ) );
                        break;
        }
        tr_write ( cur_tree );
        save();
break;

case UA:
        ua ( cur_tree );
        tr_write ( cur_tree );
        save();
break;

case UNDO:
        undo();
        tr_write ( cur_tree );
break;

case URSATZ:
        ursatz ( cur_tree );
        tr_write ( cur_tree );
        save();
break;

case WRIT:
        /*  if "write macro macroname" */
        if ( gargc == 3 && !strcmp(gargv[1],"macro") ){
                /*macro_write( gargv[2] );*/
        } else if ( gargc == 2 )
                toutput ( gargv[1] );
        else
                error("write","invalid argument" );
        tr_write ( cur_tree );
break;

default:
        error( "unknown function", gargv[0] );
break;
```

```
tree_write( fp, p )
FILE    *fp;
TNODEPTR        p;

{
TNODEPTR        q;
int             i;

if ( p-> note == SIM || p->note == SEQ ){
                                /* check for 1-line or >1-line list */
        for ( q = p->child; q != NULL && q->note != SIM && q->note != SEQ;
                                                        q = q->brother );
        indent++ ;

        if ( q == NULL )
                tr_wr_1( fp, p );
        else
                tr_wr_n( fp, p );
}else{
        fprintf( fp, "( %d %c %d )", p->accidental, p->note, p->octave );
        fprintf( fp, "\n" );
        for ( i = 0; i < indent; i++ )
                fprintf (fp, "   " );
}

}

tr_wr_1( fp, p )
FILE    *fp;
TNODEPTR        p;

{
int     i;

if ( p->note == SIM ){
        fprintf( fp, "%s", "( sim ");
        for ( p = p->child; p != NULL; p = p->brother)
                fprintf( fp, "( %d %c %d )", p->accidental, p->note, p->octave );
}else if ( p->note == SEQ ){
        fprintf( fp, "%s", "( seq ");
        for ( p = p->child; p != NULL; p = p->brother)
                fprintf( fp, "( %d %c %d )", p->accidental, p->note, p->octave );
}

fprintf( fp, ")" );
indent--;
fprintf( fp, "\n" );
for ( i = 0; i < indent; i++ )
        fprintf (fp, "   " );
}

tr_wr_n( fp, p )
FILE    *fp;
TNODEPTR        p;

{
TNODEPTR        q;
int     i;

if ( p->note == SIM ){
        fprintf( fp, "%s", "( sim ");
        fprintf( fp, "\n" );
        for ( i = 0; i < indent; i++ )
                fprintf (fp, "   " );
}else if ( p->note == SEQ ){
```

```
    fprintf( fp, "\n" );
    for ( i = 0; i < indent; i++ )
            fprintf (fp, "   " );


( p->child){
    tree_write( fp, p->child );
    q = p->child;
    while ( q->brother ){
            tree_write( fp, q->brother);
            q = q->brother;
    }
    indent--;
    fprintf (fp, "\b\b\b)\n" );
    for ( i - 0; i < indent; i++ )
            fprintf (fp, "   " );
```